VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Android aplikace pro generování síťového provozu

Android Application for Generating Network Traffic

Bc. Ondřej Freisler

Diplomová práce

Vedoucí práce: Ing. Lukáš Kapičák

Ostrava, 2021

Abstrakt

S rozvojem mobilních technologií nacházejí mobilní zařízení uplatnění v nových odvětvích, což platí i v oblasti síťového provozu, kde mohou tato zařízení nahradit běžné počítače díky bezdrátovým sítím.

Cílem této diplomové práce bylo vyvinout aplikaci pro mobilní zařízení s operačním systémem Android, která umožňuje generování síťového provozu pro protokoly HTTP, FTP, SIP, IMAP, POP3 a SMTP. Výsledný proces tohoto generování si však neklade za cíl simulovat zátěž v počítačových sítích pro vylepšení kvality služeb, jako je tomu u ostatních nástrojů podobného typu, nýbrž se zaměřuje na oblast počítačové bezpečnosti. Prostřednictvím generování síťového provozu se vyvinutá aplikace snaží simulovat chování běžného uživatele v internetovém prostředí. Smyslem tohoto chování je pak upozornit potencionálního útočníka v síti na nešifrovaný obsah síťového provozu, který byl vygenerován právě touto aplikací.

Klíčová slova

Generování síťového provozu; Android; LDAP; virtuální privátní síť; bezpečnost bezdrátových sítí; nešifrovaná komunikace

Abstract

With the development of mobile technologies, mobile devices find uses in new industries, as well as in the field of network traffic, where these devices can replace ordinary computers thanks to wireless networks.

The aim of this thesis was to develop an application for mobile devices with the Android operating system, which allows the generation of network traffic for HTTP, FTP, SIP, IMAP, POP3 and SMTP protocols. However, the resulting process of this generation does not aim to simulate the load on computer networks to improve the quality of service, as with other tools of a similar type, but focuses on computer security. By generating network traffic, the developed application tries to simulate the behavior of a normal user in an internet environment. The purpose of this behavior is then to alert a potential attacker on the network to the unencrypted content of network traffic that was generated by this application.

Key words

Generating network traffic; Android; LDAP; virtual private network; security of wireless networks; unencrypted communication

Poděkování

Rád bych poděkoval panu **Ing. Lukáši Kapičákovi** za odbornou pomoc a konzultaci při vytváření této diplomové práce.

Obsah

Seznam použitých zkratek

Zkratka	Význam
AD	Active Directory
API	Application Programming Interface
APK	Android Application Package
ARP	Address Resolution Protocol
ART	Android Runtime
ASCII	American Standard Code for Information Interchange
ATM	Asynchronous Transfer Mode
DEX	Dalvik EXecutable
DNS	Domain Name System
DVM	Dalvik Virtual Machine
FDDI	Fiber Distributed Data Interface
FTP	File Transfer Protocol
GUI	Graphic User Interface
HAL	Hardware Abstraction Layer
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IMAP	Internet Message Access Protocol
IP	Internet Protocol
IPSec	Internet Protocol Security
ISO	International Organization for Standardization
ITU	International Telecommunication Union
JPEG	Joint Photographic Experts Group
L2TP	Layer 2 Tunneling Protocol
LDAP	Lightweight Directory Access Protocol
LLC	Logical Link Control
MAC	Media Access Control

MIME	Mulitpurpose Internet Mail Extensions
MPEG	Moving Picture Experts Group
NDP	Neighbor Discovery Protocol
NetBIOS	Network Basic Input Output System
NFS	Network File System
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
PDU	Protocol Data Unit
PHP	Hypertext Preprocessor
POP3	Post Office Protocol version 3
PPP	Point-to-Point Protocol
PPTP	Point-to-Point Tunneling Protocol
QoS	Quality of Service
RIP	Routing Information Protocol
RPC	Remote Procedure Call
SAP	Service Advertising Protocol
SASL	Simple Authentication and Security Layer
SDK	Software Development Kit
SIP	Session Initiation Protocol
SMB	Server Message Block
SMTP	Simple Mail Transfer Protocol
SQL	Structured Query Language
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
URL	Uniform Resource Locator
USB-C	Universal Serial Bus Type-C
VoIP	Voice over Internet Protocol
VPN	Virtual private network
WebDAV	Web Distributed Authoring and Versioning
XML	Extensible Markup Language

Seznam ilustrací a tabulek

Úvod

Počítačová kriminalita je v dnešní době velmi známým a bohužel i rozšířeným pojmem. IT správci mnoha organizací nejsou schopni identifikovat veškerý síťový provoz, přenášející se sítěmi, které mají pod svou správou. Rozhodně by jim nemělo být lhostejné toto téma přehlížet, neboť bezpečnost samotné síťové infrastruktury je jednou z důležitých vlastností úspěšné a prosperující organizace.

Typickou situací, kdy se sítí přenáší velké množství dat, které není správce sítě schopen identifikovat, může být zejména v souvislosti s bezdrátovými sítěmi Wi-Fi, které jsou v dnešní době dostupné ve většině, ne-li ve všech organizacích. Tyto sítě jsou rovněž implementovány na veřejná místa, jako jsou obchodní centra, restaurace, školy, atd. Všechna tato místa navštíví za běžných podmínek velký počet lidí, který se odvíjí od různých faktorů (například frekventovanosti). Čím více lidí toto místo navštíví, tím existuje větší pravděpodobnost využití lokálně dostupné bezdrátové sítě. Na této síti pak provádí každý člověk svou běžnou činnost, za kterou lze obecně považovat například surfování po internetu, sledování videí, poslech hudby, ale taktéž komunikace pomocí sociálních služeb nebo přihlašování k internetovému bankovnictví. Mezi tímto velkým počtem lidí se však může často vyskytovat i útočník, který se k dané síti nepřipojil kvůli výše zmíněným aktivitám, ale proto, aby tuto síť monitoroval. Pokud získá přístup k zařízení, které tuto možnost monitorovat síťový provoz má, naskýtá se mu příležitost dostat se k velkému množství, a to i citlivých, dat.

Většina dat, která se na sítích v dnešní době přenáší, jsou nicméně šifrovaná. Existují však stále i webové servery a služby, které šifrovanou komunikaci nepodporují. Pokud se útočník dostane k těmto nešifrovaným datům, většinou neváhá a zneužije je pro vlastní potřebu. Pokud se například jedná o přihlašovací údaje, využije je pro vlastní přihlášení se ke konkrétnímu serveru.

Aplikace, která je v této práci dále vyvíjena, se zaměřuje právě na výše zmíněnou problematiku tím, že simuluje chování běžného uživatele v prostředí internetu generováním síťového provozu, v rámci nějž rovněž využívá nešifrovaných přihlašovacích údajů s cílem nalákat na ně potencionálního útočníka.

Teoretická část uvádí čtenáře do oblasti sítí, ve které nejprve definuje samotný pojem síťový provoz. Jelikož se jedná o jedno z klíčových slovních spojení celé této práce, je rovněž nutné znát obecný kontext, pod který tento pojem spadá. Síťovým provozem lze totiž z praktického hlediska nazvat množství dat, které prochází sítí v daném okamžiku, a právě proto je tento pojem v určité formě předmětem mnoha příbuzných témat. Jedná se například o parametry počítačových sítí nebo referenční model OSI, respektive TCP/IP, o kterých se práce taktéž zmiňuje.

Dalším ze základních prvků této práce je operační systém Android, neboť právě pro tento systém byla později vytvořena aplikace, umožňující generovat již zmíněný síťový provoz. Po charakteristice tohoto operačního systému je rovněž věnován prostor kapitole, která řeší otázky programování mobilních aplikací jako celku. Součástí této kapitoly je rovněž stručný popis vývojového prostředí Android Studio, ve kterém byla aplikace vyvíjena.

Následující část již pojednává o samotné aplikaci, která byla v rámci této práce vyvinuta a rovněž zdokumentována. Obsahem této části jsou nejprve témata, zabývající se smyslem vývoje, uplatněním v praxi nebo principem fungování aplikace.

Proces generování síťového provozu je poměrně širokým pojmem, díky kterému existují různé možnosti, jak jej realizovat v praxi. Až po pochopení specifického zaměření práce se smysl generování síťového provozu ubírá především do oblasti bezpečnosti v počítačových sítích, jak již bylo uvedeno dříve.

Dle zadání práce má být aplikace schopna kromě generování síťového provozu pro protokoly HTTP, FTP, SIP, IMAP, POP3 a SMTP nejprve implementovat podporu pro virtuální privátní sítě (VPN) a dále pro adresářové servery (LDAP). K adresářovému serveru se aplikace připojuje právě pomocí VPN a tento adresářový server hraje klíčovou roli právě z hlediska přihlašovacích údajů, které aplikace distribuuje po síti v nešifrované podobě.

Aplikace, jejíž chování bylo právě popsáno, byla v této práci vyvinuta, zdokumentována a porovnána s dalšími nástroji, generujícími síťový provoz. Rovněž došlo k vyhodnocení výsledků, a to jednak prostřednictvím obrázků, demonstrujících vygenerovaný síťový provoz v programu Wireshark a jednak závěrem, který tuto práci shrnuje a poukazuje na další možnosti jejího vylepšení či optimalizace.

1 Rešerše v oblasti síťového provozu

Lidstvo by se bez internetu v dnešní moderní době už zřejmě neobešlo. Můžeme jen polemizovat o tom, jak by se vyvíjela civilizace bez on-line připojení. Denně se přenáší po síti obrovské množství dat a je důležité si uvědomit, že tato čísla stále stoupají – nejen díky tomu, že tento prostředek využívá stále více lidí, ale rovněž díky novějším a pokročilejším technologiím, kdy skrze internet komunikují oproti minulosti i další zařízení.

S každým dalším zařízením, využívajícím sdílené síťové prostředky sítě pro svou vlastní potřebu, jsou kladeny na stabilitu sítě větší nároky. Celková maximální přenosová rychlost sítě se tím pádem musí dělit mezi více zařízení, což může některá negativně ovlivnit. Dalším velmi důležitým tématem, souvisejícím s využíváním síťových prostředků, je pak bezpečnost sítě samotné. Pokud se podíváme do podnikové či mezinárodní sféry, měření kvality služeb v síti je obdobně jako její bezpečnost kritickou vlastností. Dojde-li totiž k přetížení a následnému výpadku sítě například bankovní společnosti, bude to mít negativní dopad jednak pro klienty, kteří nebudou moci při výpadku sítě provádět on-line platby, tak rovněž pro samotnou společnost, kterou bude tento neočekávaný stav stát nemalé peníze a ztrátu reputace. Konkrétním klíčovým službám či zařízením tedy musí být v síti zajištěna dostatečná šířka pásma za jakýchkoliv podmínek tak, aby mohly vždy spolehlivě fungovat. Tuto problematiku řeší v informatice termín QoS, který patří mezi důležité parametry sítě.

Mezi další klíčové parametry sítě můžeme dále zařadit propustnost, rychlost přenosu dat, zpoždění, vytížení linky nebo typ přenosového média, od kterého se odvíjí další specifické proměnné včetně bezpečnosti sítě jako celku. O těchto parametrech se v této diplomové práci lze detailněji dočíst v dalších kapitolách. Další významnou úlohu hrají použité síťové protokoly a standardy. Všechna tato kritéria spadají pod společný termín, zvaný síťový provoz, jež je hlavním předmětem této práce.

1.1 Definice síťového provozu

Ve světě IT sítí chceme obvykle navrhnout a implementovat síť a serverovou infrastrukturu takovou, aby použitý hardware i software umožňoval v ideálním případě přenos dat neomezeného objemu. K tomuto scénáři slouží aplikace, umožňující simulovat generování síťového provozu, díky čemuž lze následně vyhodnotit výkon a odolnost systému, respektive sítě.

Síťový provoz by se dal definovat jako množství dat procházejících sítí v daném okamžiku, čímž je ovlivněno aktuální zatížení sítě. Efektivní organizace síťového provozu pak pomáhá při zajišťování kvality služeb v dané síti. Různé topologie sítě mohou být například implementovány pouze na základě známého či předpokládaného množství síťového provozu v systému.

[1][2]

1.2 Modely síťového provozu

Při teoretických úvahách, souvisejících s návrhem generátoru síťového provozu, musí vyvstat otázky následujícího typu: *"Jak vlastně chci tento provoz generovat? Z jakých dat mám vycházet?"* Odpovědí na tyto otázky jsou právě modely – jakési matematické podklady pro simulaci síťového provozu.

Správci sítě se obvykle při kalkulaci maximální síťové zátěže a řešení potíží, způsobených překročením této hodnoty, spoléhají zejména na specifikaci síťových prvků a dokumentaci jejich výrobců. Reálný svět ovšem ne vždy funguje tak, jak se od něj očekává. Je důležité analyzovat, jak síťová infrastruktura reaguje na zatížení při normálním provozu i v případě extrémní zátěže. Existují nicméně komplikace, díky nimž takováto analýza není triviální úlohou z hlediska implementace. Typy síťového provozu jsou v dnešních podnikových sítích velice různorodé, neboť každý typ provozu ovlivňuje síť poněkud jiným způsobem. A protože každá síť je používána jedinečným způsobem s jedinečným síťovým provozem, musí být posuzována na základě jejího vlastního typu využití. Nejlepším způsobem určení kapacity sítě a konkrétních bodů, ve kterých může dojít k selhání, je tedy ve skutečnosti generovat provoz podle modelů, jak uvádí například [3].

Modely síťového provozu jsou dnes důležitým tématem na poli IT. Jsou nezbytné pro poskytovatele internetu z důvodu správy výkonu sítě a zajišťují mimo jiné také zabezpečení sítě, kde řeší kupříkladu narušení bezpečnosti či kvalitu služeb QoS.

Správná klasifikace modelů síťového provozu usnadňuje práci správci sítě, který pak v síti jednodušeji identifikuje příčiny možných problémů. Modely síťového provozu můžeme rozlišit v závislosti na vlastnostech použitých datových bloků (paketů).

1.2.1 Model založený na portu

Jedná se o nejjednodušeji a nejrychleji rozpoznatelnou metodu klasifikace paketů síťového provozu a jako taková je široce využívána především díky tomu, že aplikace mají často pevně přidělená statická čísla portů. Tyto porty jsou registrovány pod organizací IANA.

Mnoho *peer-to-peer* aplikací (např. multimédia, videohry atd.) nicméně fixní čísla portů nemají a namísto toho využívají dynamická čísla portů často využívaných protokolů, která nejsou registrována pod IANA a díky kterým tak přináší tato klasifikace mnohdy nepřesné výsledky.

1.2.2 Model založený na užitečném zatížení

Tento model je nazýván také jako *Deep Packet Inspection* a využívá algoritmů, které na základě analýzy obsahu paketů identifikují typ síťového provozu. Obdobně jako u modelů, založených na portu, tato metoda často nedokáže korektně identifikovat typ síťového provozu obzvláště u aplikací, využívajících šifrovaný tok dat. Mezi další nevýhody by se mohla uvést nutnost aktualizace vyhledávacích algoritmů v případě nových aplikací.

1.2.3 Model založený na strojovém učení

Metody založené na strojovém učení (*Machine Learning*) používají techniky strojového učení k nalezení známých posloupností dat v paketech. Používají se zde 2 techniky:

- Učení pod dozorem (*supervised learning*) – Na základě známých vstupních a výstupních dat se predikují budoucí výstupní data pomocí technik zvaných klasifikace či regresní analýza dat.
- Učení bez dozoru (*unsupervised learning*) – Používá se k odvození závěrů, kdy se ze vstupních dat hledají skryté vzory nebo struktury pomocí shlukové analýzy.

[3][4]

1.3 Metody zisku vzorků dat

Abychom mohli správně charakterizovat síťový provoz a vytvořit tak odpovídající konkrétní model, je nutné analyzovat provoz reálné sítě, abychom měli z čeho vycházet. Takovýto rozbor je obvykle možný provést ze získaných vzorků síťového provozu, který probíhal po určitý čas. Následuje popis metod, které jsou používány při pokusu o charakterizaci internetových aplikací – tedy i síťového provozu.

1.3.1 Zisk dat z logů

Většina webových serverů uchovává záznamy o komunikacích (tzv. logy) se soubory, které poskytly, a to z důvodů od monitorování provozu až po shromažďování demografických údajů o uživatelích. Model pracovní zátěže lze vyhotovit zpracováním jednak logů serverových, tak klientských. Serverové logy obsahují dotazy všech uživatelů na danou webovou stránku a jejich nevýhodou je skutečnost, že nemohou jednoduše zachytit aktivitu uživatelů napříč více webovými stránkami. Metoda sběru dat z logů klienta zase zachycuje aktivitu uživatelů mezi více webovými stránkami mnohem lépe než sběr dat z logů serveru.

1.3.2 Zisk dat z nástrojů pro analýzu paketů

Další metodou analýzy síťového provozu je použití nástrojů (ať už hardwarových či softwarových), jejichž primárním účelem je analýza a monitorování konkrétních síťových rozhraní. Tyto nástroje pomáhají identifikovat, klasifikovat a řešit potíže se síťovým provozem ať už podle typu aplikace nebo na základě zdrojové/cílové IP adresy. Na trhu existuje takovýchto nástrojů řada, z nichž většina se pro splnění svého účelu spoléhá na svá aplikační rozhraní, známá jako *libpcap* (pro *unixové systémy*) nebo *WinPcap* (pro *Windows*).

Tyto nástroje fungují na principu zachytávání paketů. To znamená, že zachycují síťový provoz, procházející kabelovou, optickou či bezdrátovou sítí, a kopírují je do speciálního souboru ve formátu *PCAP*. Z těchto souborů lze analyzovat dříve proběhlý síťový provoz a pro implementaci aplikace, která z něj dokáže převzít data, je možné je na základě jejich charakteristiky a povahy nastavit jako vstup při generování síťového provozu. Pakety, které nejsou adresované danému zařízení je možné zachytávat také, nicméně síťová karta musí být nastavena do režimu monitorování.

[5][6]

1.3.3 Port mirroring

Provoz na síti se dá rovněž analyzovat prostřednictvím zrcadlení portů - tzv. port mirroringu, který je realizován na síťových switchích. Díky této metodě může přepínač posílat kopie síťových paketů mezi zařízeními prostřednictvím svých portů, kdy zdrojový port pro zařízení A přidělí cílovému portu pro zařízení B. Existuje jak lokální zrcadlení portů, kde všechny zdrojové porty jsou umístěny na stejném síťovém zařízení jako cílové porty, tak vzdálené, kde zdrojové a cílové porty se nevyskytují na stejném zařízení. Kromě analýzy síťového provozu slouží tato metoda rovněž k diagnostice chyb nebo se využívá jako ladící nástroj.

[7]

1.4 Parametry generování síťového provozu

Při generování síťového provozu je nezbytné rozumět parametrům, které tento proces charakterizují. Mezi tyto parametry jednoznačně patří zejména cílová IP adresa, na kterou bude generování směřovat, dále objem dat a v neposlední řadě také časový údaj, jak dlouho má generování probíhat. Pomocí známého objemu dat a časového údaje délky jeho generování je možné jednoduše stanovit frekvenci generování.

Protokoly TCP/UDP transportní vrstvy modelu OSI a dále protokoly aplikační vrstvy pak tvoří další důležitý parametr, související s generováním síťového provozu. Použitému protokolu následně odpovídá různá charakteristika dat.

1.5 Parametry sítě

Kromě parametrů síťového provozu jako takového je nutností vzít v potaz i obecné parametry samotné sítě, ve které tento síťový provoz probíhá, neboť i na nich je výsledná charakteristika toku dat závislá. S těmito parametry je žádoucí uvědomit si rozdíl mezi sítí ideální a reálnou. V ideální síti neexistují zpoždění či chyby, přenosová rychlost je teoreticky ovlivněná jen parametry sítě a zařízení mezi sebou komunikují přímo. V praxi to tak ale rozhodně nefunguje, neboť každou počítačovou síť ovlivňují především následující parametry:

- **Typ přenosového média** – Řeší se na fyzické vrstvě modelu OSI. Jeho úkolem je vytvoření cesty mezi zařízeními tak, aby bylo možné vysílat signály (data). Přenosová média mohou být jak metalické kabely (kroucená dvojlinka, koaxiální kabel), tak bezdrátové přenosy, kdy hovoříme o rádiových, mikrovlnných, infračervených a optických komunikacích [9].
- **Šířka pásma (bandwidth)** – Jedná se o maximální množství dat, které je možné přenést sítí za jednotku času neboli maximální propustnost počítačové sítě. Definuje přenosovou rychlost.
- **Přenosová rychlost** – Udává se jako počet bitů, přenesených za jednotku času. Při vyčíslení větších rychlostí se využívá předpon *Mezinárodního systému jednotek SI*.
- **QoS** – Schopnost sítě nastavit určitému kritickému síťovému provozu (popř. zařízení) vyšší prioritu využitím větší šířky pásma k jeho spolehlivé funkčnosti tak, aniž by byl úplně omezen ostatní síťový provoz, který využívá určitou šířku pásma také. Mezi příklady provozu s vysokou prioritou, který je potřeba spolehlivě zajistit, patří kupříkladu VoIP nebo online videohry, kde je nutné pracovat v reálném čase. QoS je většinou spravováno pomocí funkcí, zabudovaných do firmwaru routeru nebo serveru, který řídí ostatní síťové prvky. Je ovšem nezbytné, aby veškeré tyto síťové prvky podporovaly schopnost QoS, jinak budou operovat na principu služby *best-effort*, která se sice snaží o „nejlepší“ způsob doručení dat, nicméně počítá s aktuálním zatížením sítě. Generování síťového provozu hraje při testování QoS zásadní roli [10][11].
- **Zpoždění (delay)** - Doba přesunu bitových dat z jednoho koncového bodu do druhého. Bývá způsobeno frontami v síťových prvcích (tzv. buffering) a závisí také na době síření signálu. Často je také označováno jako latence.
- **Rozptyl zpoždění (jitter)** - Rozdíl mezi zpožděním předávání dvou po sobě jdoucích přijatých paketů v určitém toku dat. To znamená, že při přenosu paketů po síti dochází ke kolísání zpoždění, čímž se zhoršuje kvalita služby [12].

- **Ztrátovost paketů (packet loss)** – Pakety se při cestě sítí ztrácí především kvůli fyzickým vlastnostem přenosového média, přetížením procesoru síťových prvků nebo přeplněním jejich front.
- **Vytížení sítě** – U páteřních sítí se považuje za kritické využití kapacity již hodnota kolem 80 %, a proto je nutné brát tento parametr v potaz, aby nedocházelo k pozastavení veškerého síťového provozu či jeho úplnému výpadku. To se řeší optimalizací směrování síťového provozu nebo zvýšením kapacity.

[8]

1.6 Referenční model OSI a zapouzdření

Při zaměření této práce na problematiku sítí je důležité zmínit se o referenčním modelu OSI. Většina výrobců zařízení, která se využívají v informačních a telekomunikačních systémech, totiž popisuje své produkty právě podle vztahu k tomuto modelu. Byl vytvořen *Mezinárodní organizací pro normalizaci* (https://www.iso.org/), zkráceně ISO.

Účelem tohoto referenčního modelu je stanovení pravidel, podle kterých se internetové aplikace mohou dorozumět v síti a vyměňovat si tak data. Referenční model OSI se skládá ze sedmi vrstev, z nichž každá je zodpovědná za provádění dílčích úkolů, týkajících se konečného odesílání a přijímání dat. Vrstvy mezi sebou komunikují pouze sousední a to tak, že při vysílání dat ze zdrojového zařízení se začíná vrstvou nejvyšší (sedmou) směrem k nejnižší. Následuje jejich přenos sítí a následně, když se data dostanou k cílovému zařízení, směřují pro změnu od nejnižší vrstvy po nejvyšší. Komunikaci mezi jednotlivými vrstvami popisují komunikační protokoly a výměna informací probíhá formou **protokolových datových jednotek** (PDU).

V souvislosti s referenčním modelem OSI se často uvádí, že se jedná o proces zapouzdření, což odkazuje na postupné přidávání informací od každé konkrétní vrstvy k datům tak, jak prostupují modelem.

[13][14]

1.6.1 Fyzická vrstva

Nejnižší vrstva modelu OSI zajišťuje přenos signálu daným prostředím (elektrickým, optickým či vzduchem) a defacto určuje, jak jsou jednotlivé bity reprezentovány v signálech, přenášených těmito prostředími. Dále poskytuje hardwarové prostředky pro odesílání a přijímání dat na nosiči, definuje kabely, síťové karty a jejich fyzické aspekty. PDU je zde **bit**. Do této vrstvy spadají např. standardy: USB, Bluetooth, IEEE 802.11, RS232, RJ45

1.6.2 Spojová vrstva

Spojová vrstva je rozdělena do dvou dílčích vrstev – MAC a LLC, které řeší fyzické adresování. Celkově se stará se o přenos dat přes lokální přenosové médium. Toho docílí pomocí metod řízení přístupu k médiu, což řídí vrstva MAC. Vrstva LLC má kromě synchronizace rámců pomocí návěstí (*flag*) také na starost kontrolu chyb. PDU této vrstvy je nazývána **rámec**. Druhou vrstvu reprezentují protokoly: Ethernet, Token Ring, PPP, Frame Relay, ATM, ARP, NDP, FDDI.

1.6.3 Síťová vrstva

Primární funkcí třetí vrstvy je přenos dat sítí a logická adresace, čehož protokoly této vrstvy docílí zapouzdřením dat s informacemi o IP adresách a výběrem vhodných síťových tras (směrování).

PDU se zde nazývá **paket**. Každému paketu je přidělena IP hlavička, která obsahuje mimo jiné informace o IP verzi, délce paketu, zdrojové a cílové IP adrese. Mezi protokoly této vrstvy patří IP, IPSec, ICMP, RIP, OSPF.

1.6.4 Transportní vrstva

Než mohou mezi sebou zařízení posílat data, je nutností mezi nimi nejprve navázat spojení. Čtvrtá vrstva zajišťuje kompletní přenos dat mezi koncovými zařízeními a je zodpovědná za řízení toku dat a kontrolu chyb. Určuje dále kolik dat se má odeslat a jakou rychlostí. Je rovněž schopná identifikovat aplikace pomocí portů. Mezi protokoly této vrstvy patří TCP a UDP. PDU této vrstvy se značí jako **segment** (v případě TCP protokolu), respektive **datagram** (pro UDP).

Protokol TCP zaručuje, že vyslaná data dojdou k cíli ve stejném pořadí, jako byla odeslána, a to tak, že pokud se ztratí, jsou poslána znovu (jsou spolehlivá). Kvůli tomuto trasování komunikačních toků nicméně TCP spotřebovává více výpočetních zdrojů zařízení, což je promítnuto i na nižší přenosové rychlosti oproti UDP. TCP provoz je proto charakteristický především pro aplikace a služby, které vyžadují bezpečnost a jistotu doručení.

Oproti tomu protokol UDP nezaručuje, že vyslaná data dorazí do cíle a také nepočítá s tím, že datagramy dojdou ve stejném pořadí. Tyto vlastnosti požadují zejména aplikace a služby, které preferují rychlost přenosu dat na úkor kvality. Příkladem je oblast IP telefonie či streamování videa a hudby.

1.6.5 Relační, prezentační a aplikační vrstva

Tyto vrstvy již nejsou z pohledu problematiky generování síťového provozu důležité, neboť manipulaci se síťovým provozem řeší zejména vrstvy předchozí. Není nicméně od věci alespoň stručně uvést jejich účel v rámci modelu OSI.

Relační vrstva koordinuje a ověřuje konverzace mezi aplikacemi, stará se o opětovné připojení nebo určuje dobu, po kterou má systém čekat na odpověď jiné aplikace. Definuje sokety (zřízení relace). Zástupci: NetBIOS, RPC, SAP, SQL.

Prezentační vrstva se věnuje formátování dat na základě sémantiky nebo syntaxe cílové aplikace, pro kterou jsou tyto data určena. Data se zde také šifrují a dešifrují. Zástupci: ASCII, JPEG, MPEG, MIME, SSL, TLS.

Aplikační vrstva umožňuje uživateli komunikovat po síti skrze síťovou aplikaci nebo službu. Veškeré služby, které tato vrstva poskytuje, se liší od konkrétní aplikace – přenos souborů, e-mail, ochrana soukromí, ověření uživatele. Protokolů této vrstvy existuje mnoho - např. DNS, FTP, HTTP, NFS, POP, SMTP, Telnet.

[15][16][17]

1.7 Referenční model TCP/IP

Kromě modelu OSI stanoví pravidla internetových aplikací a popisuje veškerou síťovou komunikaci rovněž referenční model TCP/IP, jenž se skládá ze čtyř vrstev.

Každý odeslaný e-mail, otevřená webová stránka či sdílený soubor jsou distribuovány po internetu jako tisíce malých bloků, známých jako datové pakety. Tyto pakety jsou přenášeny

prostřednictvím rodiny protokolů TCP/IP, které se dělí do čtyř vrstev: vrstva síťového rozhraní, síťová, transportní a aplikační.

Každý paket prochází aplikační vrstvou do vrstvy TCP (transportní), kde má přiděleno číslo portu. Následně paket migruje na IP vrstvu (síťovou), kde je mu přidělena cílová IP adresa. Jakmile má paket přidělen port a cílovou adresu, je možné ho odeslat přes internet. Odeslání se provádí prostřednictvím vrstvy síťového rozhraní, která převádí paketová data na signály. Jakmile dorazí paket na místo určení, data použitá ke směrování paketu se odstraní a následuje opětovný průchod rodinou protokolů TCP/IP, nicméně od nejnižší vrstvy po nejvyšší. Paket je opět v původní podobě, v jaké byl před odesláním, po průchodu nejvyšší vrstvou.

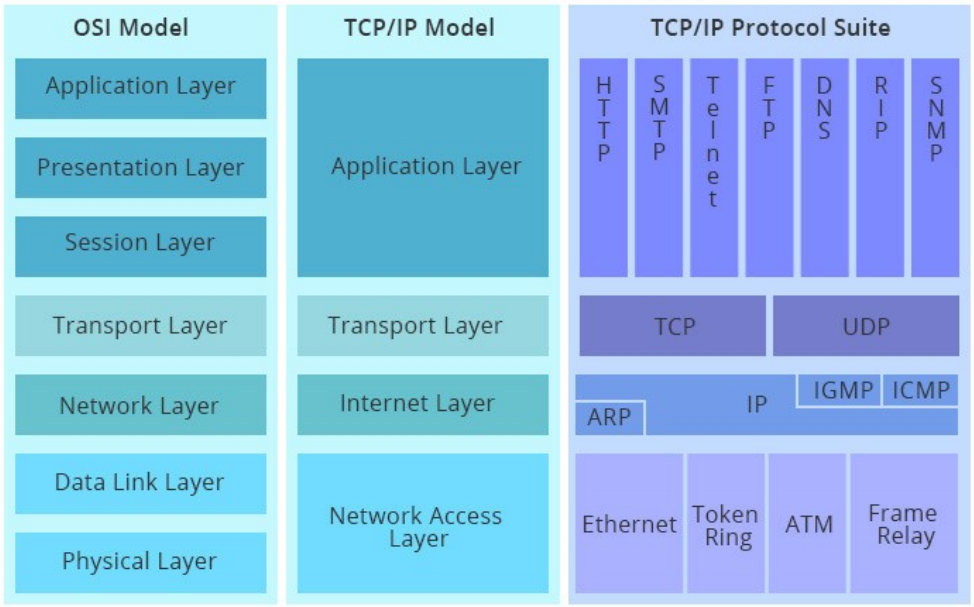
[17]

1.8 Porovnání referenčních modelů OSI a TCP/IP

Vzhledem k významům obou referenčních modelů je model OSI pouze modelem konceptuálním. To znamená, že se používá především k popisu, diskusi a pochopení jednotlivých síťových funkcí. Model TCP/IP je však navržen tak, aby řešil konkrétní soubor problémů, a nikoliv aby sloužil pouze k obecnému popisu veškeré síťové komunikace. Používá se dále k modelování aktuální internetové architektury a k poskytování sady pravidel, která jsou dodržována všemi formami přenosu po síti.

Přestože je model OSI více obecný, většina protokolů a systémů dodržuje právě jej. Naproti tomu TCP/IP model je založen na standardních protokolech, souvisejících s vývojem *Internetu*, jakožto celosvětové informační a komunikační sítě. Další věcí, kterou charakterizuje OSI model je pak skutečnost, že jednodušší aplikace striktně nevyužívají všech sedmi vrstev – zatímco nejnižší 3 vrstvy jsou povinné pro jakoukoliv datovou komunikaci, namísto obvyklých horních třech vrstev může aplikace využít jedinečnou vrstvu rozhraní.

[17]



Obrázek 1.1: *Uspořádání vrstev modelů OSI a TCP/IP s odpovídajícími protokoly* [17]

Z obrázku 1.1 lze vypozorovat rozdílné uspořádání jednotlivých vrstev referenčních modelů OSI a TCP/IP včetně uvedení protokolů, vztahujících se k vrstvám modelu TCP/IP.

2 Analýza operačního systému Android

Jelikož je tato diplomová práce zaměřena v první řadě na generování síťových prostředků pro operační systém (dále OS) Android, je nyní vhodné věnovat podstatnou část teorie právě tomuto operačnímu systému. Nyní následuje popis základní charakteristiky tohoto mobilního OS a seznámení se s důležitými body jeho historie včetně původu vzniku oficiálního loga Android. Další část je věnována popisu jednotlivých verzí tohoto OS, kterou následuje popis architektury včetně jednotlivých vrstev.

2.1 Základní charakteristika OS Android

Pod dnes již hojně rozšířeným a používaným pojmem *Android* si lze představit mobilní operační systém, který je založený na modifikované verzi jádra systému *Linux* a dalším open source softwaru (spadajícím pod pojem *Android Open Source Project*). Tento software je psán pomocí vývojových nástrojů, obecně známých jako *Android SDK*. Pokaždé, kdy tedy vyjde nová verze OS *Android*, vyjde také odpovídající SDK sada. Aby mohli vývojáři psát programy s nejnovějšími funkcemi, musí si také nainstalovat odpovídající nově implementovanou sadu SDK. Každá takováto sada zahrnuje následující komponenty:

- knihovny,
- debugger,
- emulátor,
- aplikační rozhraní (API) včetně dokumentace,
- ukázky zdrojových kódů s tutoriály (výukovými programy).

Více k tématu programování je uvedeno v kapitole "Programování mobilních aplikací".

[18]

Android je vyvíjen konsorciem vývojářů, známých pod pojmem *Open Handset Alliance*, které sponzoruje světový gigant *Google*. Právě díky chytrým telefonům, tabletům a jim podobným zařízením s OS Android si lidé snaží usnadnit životy. Od efektivního využití GPS navigace pro úsporu času, přes uživatelsky přívětivou komunikaci s přáteli, až po sledování srdečního tepu na chytrých hodinkách, to jsou jen úlomky z velmi mnoha dalších pozitivních vlastností, které přinášejí lidem zařízení s podporou OS Android. Existují ovšem i konkurenční mobilní platformy jako zejména *iOS* společnosti *Apple*, se kterou Android soutěží ve vytrvalé bitvě o vůdce na poli operačních systému mobilních zařízení. Android je open source platforma, což znamená, že kromě vývojářů z konsorcia *Open Handset Alliance* se může na jejím vývoji podílet také kdokoliv jiný. Zdrojový kód je licencován pod licencí *Apache*.

Co se týče bezpečnosti, uživatel OS Android se může rozhodnout, zda bude chtít svá data sdílet (například historii polohy či aktivity na webu). Pokud aplikace začne využívat polohu uživatele, zatímco ji uživatel nevyužívá, dostane notifikaci, která je další důležitou charakteristickou vlastností OS Android. Jedná se o upozornění, které se uživateli objeví na tzv. notifikační liště (stavovém řádku), která je pro uživatele zařízení jednoduše dostupná a ve které má efektivní přehled, jaké události se zrovna staly či dějí – SMS zpráva, zmeškaný hovor, přehrávání hudby, aktivní budík, aktivní stahování, aktuální stav počasí, příchozí e-mail, sociální sítě nebo právě upozornění o možném narušení bezpečnosti a mnoho dalších. Pro změnu příchozích upozornění je uživateli dovoleno měnit oprávnění všech nainstalovaných

aplikací. O této problematice je v této práci později pojednáváno při samotné implementaci aplikace pro generování síťového provozu. Spolu s notifikacemi jsou na stavovém řádku permanentně také údaje o čase, stavu baterie, či konektivitě k sítím.

Každý člověk má svůj vlastní způsob života a používání svých zařízení, a proto i OS *Android* nabízí nepřeberné množství aplikací třetích stran a nastavitelných systémových funkcí včetně upravení vzhledu, kde si každý dle svých osobních preferencí může vybrat to své. Digitální obsah, kterým se rozumí zejména aplikace, hudba, filmy či knihy je v OS *Android* dostupný přes distribuční službu *Google Play* (https://play.google.com/store), která nabízí kontrolovaný digitální obsah jak zdarma, tak i placený.

Výchozí uživatelské rozhraní je zde založeno hlavně na přímé manipulaci pomocí dotykových akcí (gest), které odpovídají akcím reálného světa – přejíždění, poklepání, uchycení objektů atp. společně s virtuální klávesnicí.

Zařízení s OS *Android* využívají domovskou obrazovku (angl. *homescreen*), což je jakási obdoba plochy s ikonami, známá z počítačových OS. Kromě ikon, představujících zástupce aplikací na domovské obrazovce, se zde vyskytují také *widgety*, které zobrazují živý, automaticky aktualizovaný obsah, jako je předpověď počasí, hodiny, e-mailová schránka atp. [20].

Android zařízení jsou v drtivé většině napájena z baterií. Ať už se jedná o hojně využívané lithium-iontové nebo novější lithium-polymerové baterie, OS *Android* spravuje své procesy tak, aby spotřeba energie byla minimální. Pokud aplikace není zrovna používána, systém pozastaví její činnost a ona tak nevyužívá energii baterie, ani zdroje výpočetního systému [21].

Správa aplikací probíhá následujícím principem: Pokud OS zjistí nedostatek paměti, automaticky začne uzavírat neaktivní procesy, počínaje těmi, které byly neaktivní nejdelší dobu [22].

V souvislosti s předmětem práce je vhodné se rovněž zmínit o možnostech internetového připojení mobilních zařízení s OS *Android*, kde kromě klasického bezdrátového připojení Wi-Fi existuje také možnost jeho sdílení přes Bluetooth anebo pomocí klasického ethernetového připojení s použitím příslušného USB-C adaptéru.

[19][23]

2.2 Historie

Společnost *Android Inc.* byla založena v říjnu roku 2003 v Palo Alto v Kalifornii, tedy dávno předtím, než se celosvětově uchytil pojem *smartphone* a než *Apple* oznámil své první zařízení s *iOS*. Jejími čtyřmi zakladateli byli Rich Miner, Nick Sears, Chris White a Andy Rubin.

Spolčenost vzbudila zájem investorů poté, kdy v roce 2004 demonstrovala, jak se OS *Android* nainstalovaný na kameře bezdrátově připojuje k počítači. Tento počítač měl posléze možnost připojení se k dříve přezdívanému "*datovému centru Android*" (dnes odpovídajícímu cloudovému úložišti), na kterém by majitelé svých kamer a fotoaparátů měli možnost ukládat svá data.

Společnost zpočátku vůbec neuvažovala o vytvoření operačního systému, který by se stal dominantou mobilních výpočetních systémů, nicméně vzhledem k situaci na stagnujícím trhu s mobilními zařízeními a úspěšnou demonstrací svého systému v kamerách se rozhodla vyrábět OS pro mobilní telefony.

Důležitá událost historie společnosti *Android* se stala v roce 2005, kdy jejím novým vlastníkem se stala firma *Google*. V té době bylo rozhodnuto, že jako základ OS *Android* bude použito jádro systému Linux. Na toto rozhodnutí se také pojil důležitý bod, a to že samotný *Android* se stane open source platformou, tzn. že může být zdarma nabízen výrobcům mobilních telefonů třetích stran.

V roce 2007 uvedla společnost *Apple* na trh první *iPhone*, čímž zahájila novou éru v oblasti mobilních zařízení. Mezitím pracovala společnost *Android* na svém OS stále v tichosti, nicméně v listopadu téhož roku začala s odhalováním svých plánů pro boj s konkurencí. Právě tehdy se zformovalo konsorcium *Open Handset Alliance*, které tehdy zahrnovalo výrobce telefonů jako *HTC* a *Motorola*, výrobce čipů *Texas Instruments* či *Qualcomm* a operátory jako *T-Mobile*.

V září roku 2008 spatřil světlo světa první oficiální smartphone OS *Android* s názvem *T-Mobile G1*, který vyráběla společnost *HTC* (známý také pod označením *HTC Dream*). Disponoval vysunovacím 3,2palcovým dotykovým displejem spolu s fyzickou klávesnicí. Tomuto telefonu se nedostalo příliš pozitivních technických recenzí, nicméně ve své první verzi OS *Android 1.0* již obsahoval ochranné známky a integrované služby budoucího, dnes již můžeme říct velmi úspěšného obchodního plánu společnosti *Google*, jako jsou *YouTube*, *Gmail*, *Google Mapy*, webový prohlížeč *Chrome* a další.

[19][24][25]

2.3 Původ originálního Android loga

Můžeme se jistě shodnout, že většina dnešní moderní civilizace se již se "zeleným robotem", který je jedním z nejpopulárnějších log v technologickém světě, setkala. Toto logo bylo vytvořeno ruskou grafickou designérkou Irinou Blok, která byla zaměstnankyní Google a která uvedla, že tento design byl částečně inspirován toaletními logy, představujícími "Muže" a "Ženy". Pouze díky tomu, že systém Android je open source, je i jeho logo často modifikováno, neboť Google tyto úpravy dovoluje díky licenci *Creative Commons 3.0*.

[24][26]

Obrázek 2.1: *Logo společnosti Android* [24]

2.4 Verze OS Android

Od svého počátečního vydání roku 2008 až po dnešek se OS Android proměnil kromě vizuální stránky také funkčně a koncepčně. V tabulce 2.1, která následuje za tímto textem, jsou uvedeny verze tohoto systému v pořadí od nejstarších po nejnovější. Společně s číselným označením nese každá verze také speciální název, odpovídající sladkostem. Na téma, proč společnost *Google* zvolila právě takováto pojmenování, se na internetu vedlo již mnoho diskusí, nicméně ku příležitosti vydání verze *KitKat* uvedla:

"Android je operační systém, který využívá více než 1 miliarda mobilních telefonů a tabletů. A protože tato zařízení dělají náš život tak sladkým, je každá verze pojmenovaná právě po dezertu."

[27]

Od verze *10* se nicméně společnost rozhodla v tomto rituálu nepokračovat.

Velmi důležitým údajem zejména pro developery je u každé verze OS také udávaná verze podporovaného aplikačního rozhraní (API level), které v zásadě určuje, jak by mezi sebou jednotlivé softwarové komponenty měly spolupracovat. Kvalitní aplikační rozhraní usnadňuje samotný vývoj aplikací poskytováním stavebních bloků, které pak programátor dává dohromady. Verze API rovněž zodpovídá za programování komponent grafického uživatelského rozhraní.

Tabulka 2.1: *Přehled verzí operačního systému Android* [31][36]

Číslo verze	**Název**	**API level**	**Rok vydání**
1.0	Apple Pie	1	2008
1.1	Banana Bread	2	2009
1.5	Cupcake	3	2009
1.6	Donut	4	2009
2.0 / 2.1	Eclair	5, 6, 7	2009
2.2.x	Froyo	8	2010
2.3.x	Gingerbread	9, 10	2010
3.x.x	Honeycomb	11, 12, 13	2011
4.0.x	Ice Cream Sandwich	14, 15	2011
4.1.x /4.2.x / 4.3	Jelly Bean	16, 17, 18	2012
4.4	KitKat	19, 20	2013
5.x	Lollipop	21, 22	2014
6.0	Marshmallow	23	2015
7.0 / 7.1	Nougat	24, 25	2016
8.0 / 8.1	Oreo	26, 27	2017
9.0	Pie	28	2018
10.0	Android 10 (Q)	29	2019
11.0	Android 11	30	2020

Rozdíly mezi jednotlivými verzemi samozřejmě nejsou jen rozdílné názvy a verze podporovaných API, ale je to hlavně samotný vývoj hardwaru a nejrůznějších aplikací a funkcí, díky kterým plní mobilní zařízení v dnešní společnosti tak důležitou funkci.

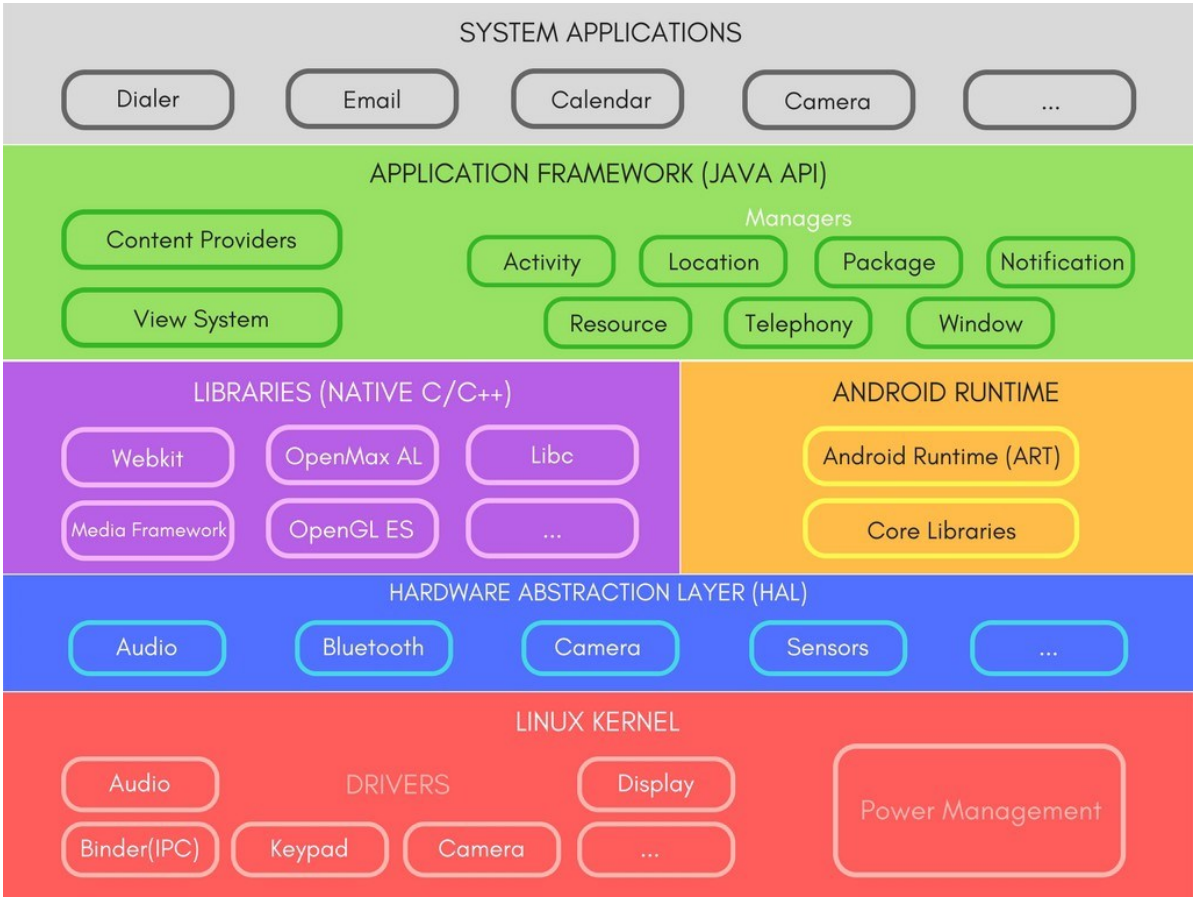
Mezi hlavní oblasti vývoje mobilních zařízení obecně by se daly zařadit parametry jako výkon, možnosti připojení, design, využití baterie, úroveň zabezpečení, kvalita fotoaparátu a mnoho jiných. Detailní analýza této oblasti by se mohla stát i hlavním předmětem jiné vysokoškolské práce.

2.5 Architektura

Společným znakem pro všechny verze OS Android je bezpochyby jeho architektura (sada komponent, které vzájemně spolupracují na splnění určitého úkolu nebo provedení funkce), která je rozdělena na šest částí, kde každá plní určitou roli. Pod těmito rolemi si lze představit například správu hardwarových ovladačů, frameworků, knihoven nebo aplikačních rozhraní, které poskytují podporu

konkrétním síťovým službám. Následovat bude nyní detailnější popis jednotlivých vrstev od té nejnižší společně s grafickou reprezentací, demonstrující uspořádání těchto vrstev.

[28]



Obrázek 2.2: *Architektura operačního systému Android* [29]

2.5.1 Linuxové jádro

Nejnižší vrstva se nazývá linuxové jádro (angl. *Linux Kernel*), které tvoří základ každého zařízení s OS Android. Toto jádro poskytuje podporu pro mnoho základních funkcí OS, mezi které patří správa zařízení, pamětí, procesů nebo zabezpečení. Jeho součástí jsou také ovladače, díky kterým dokáže Android komunikovat s hardwarovými prvky zařízení.

Vývoj tohoto jádra je doposud jedním z největších skupinových projektů na světě. Do tohoto projektu se zapojilo již více než 4000 vývojářů z více než 450 různých společností. Vydáno bylo doposud 6 vydání (*releases*). Na konci roku 2016 zabíral zdrojový kód linuxového jádra společně s jeho dokumentací a skripty na 22 milionů řádků, nicméně tento počet je společný pro všechny dostupné čipy a hardwarové ovladače. Průměrně využívá mobilní zařízení v dnešní době zhruba 3 milióny řádků kódu.

Od vydání jádra, označeného jako *2.6*, se komunita uchýlila od předchozího odděleného vývojového modelu ke společnému - tzv. *stable only* modelu. Tato změna způsobila urychlení vývoje novějších vydání a také to, že vývoj směřoval jedním konkrétním směrem (naproti více odvětvím, jako tomu bylo dříve).

[30]

2.5.2 Vrstva hardwarové abstrakce

Hardware Abstraction Layer (HAL) poskytuje standardní rozhraní, kterými vystavuje hardwarové prostředky specifických komponent mobilního zařízení vyšší vrstvě, zvané *Java API Framework*. Těmito komponenty rozumíme ku příkladu *Bluetooth* nebo kameru. Tato vrstva obsahuje knihovny, které OS načítá v případě, že si je zavolá již zmíněná vrstva *Java API Framework*.

2.5.3 Android Runtime

Před verzí OS Android 5.0 obsahovala tato vrstva virtuální stroj zvaný *Dalvik* (často jen DVM). Jedná se o speciální Java virtuální stroj, optimalizovaný pro *Android*. Umožňuje každé aplikaci pro OS *Android* spustit si svůj vlastní proces a instanci, což umožňuje spuštění více aplikací současně. Od již zmíněné verze OS Android 5.0 (včetně) nahradil DVM jeho nástupce – ART. Ten je konstruován tak, aby dokázal spustit více virtuálních strojů na zařízeních s nízkou pamětí spouštěním souborů s příponou *dex*. Jedná se o tzv. *bytekód* formát (snadno přenositelný kód), navržený speciálně pro *Android*, který je optimalizován pro minimální paměťovou stopu. Programy pro vývoj *Android* aplikací, jako je i *Android Studio*, používané v této práci, převádí zdrojové kódy z formátu *.class* právě na soubory s příponou *dex*.

2.5.4 Nativní knihovny C/C++

V této vrstvě architektury OS *Android* jsou obsaženy všechny knihovny, potřebné pro funkčnost ostatních vrstev, které potřebují k chodu svého nativního kódu příslušné nativní knihovny, psané v programovacích jazycích C a C++. Tím jsou řečeny také knihovny, založené na programovacím jazyce *Java*, které jsou potřebné pro tvorbu aplikací s OS *Android*. Aplikační rozhraní jsou poskytována aplikacím právě po zajištění funkčností těchto knihoven. Například díky knihovnám *OpenGL* je možné přistupovat k *Java OpenGL API*, čímž vzniká podpora pro kreslení s 2D/3D grafikou v konkrétní aplikaci.

2.5.5 Java API Framework

Celá sada funkcí samotného OS je vývojářům k dispozici prostřednictvím právě této vrstvy. Tato aplikační rozhraní, psaná v jazyce *Java*, jsou jakýmisi základními stavebními bloky (většinou ve formě Java tříd) pro vytváření aplikací pro platformu *Android*. Zástupci těchto rozhraní jsou zejména tzv. "správci" (*managers*), kteří zodpovídají za konkrétní logická odvětví. Následuje uvedení příkladů některých z nich:

- Správce zdrojů – Poskytuje přístup k prostředkům, jako jsou grafické soubory, řetězce, grafické rozvržení aplikace, atd.
- Správce aktivit – Má na starost životní cyklus aplikací a poskytuje tzv. *back stack* (aktivity na sebe navazují, je možné vracet se mezi nimi zpět – uživatel se nedostane do "slepé uličky").
- Správce oznámení – Stará se o stavový řádek, na kterém aplikacím uděluje možnost zobrazovat si svá oznámení.

Kromě jednotlivých správců mají v této vrstvě zastoupení také dvě další elementární rozhraní:

- Poskytovatel obsahu (*Content Provider*) - Umožňuje aplikaci přístup k datům jiných aplikací. Tento přístup může využívat například data základních aplikací, jako jsou Kontakty, Hodiny nebo Fotoaparát.

- Systém zobrazení (*View System*) - Jedná se o systém, definující grafické uživatelské rozhraní (GUI) vyvíjené aplikace. Pomocí GUI je definována struktura (rozložení) celé aplikace a také její vzhled. Využívá se zde hierarchie objektů *View* a *ViewGroup*. S těmito objekty a jejich potomky se lze dále setkat při praktickém vývoji aplikace na generování síťového provozu.

2.5.6 **Systémové aplikace**

Nejvyšší vrstvu architektury OS *Android* tvoří již samotné aplikace, které jsou výsledkem tvorby vývojářů. Základní předinstalované aplikace, které uživatel získá s koupí nového mobilního zařízení, respektive s novým OS *Android*, poskytují danému zařízení základní funkčnost. Jedná se o aplikace typu kontakty, SMS, kalendář, hodiny, email atd.

Uživatel má nicméně svobodnou volbu, zda bude chtít využívat předinstalovaných systémových aplikací, které nemají žádný speciální účel pro funkčnost OS a mohou být bez obav nahrazeny, popř. jestli si stáhne jinou, preferovanější aplikaci z třetí strany.

Další tématikou z hlediska systémových aplikací jsou pak aplikace, poskytované společností *Google*. Některá zařízení mají tyto aplikace již nainstalované v továrním nastavení v závislosti na smlouvě mezi jejich výrobci a touto společností.

[28][29]

3 Programování mobilních aplikací

Programování mobilních aplikací již přímo souvisí s vývojem aplikace, která je později v této práci vypracována a zdokumentována. Proto je nyní na místě seznámit se s detaily, týkající se oblasti programování a také s prostředky, ve kterých vývoj aplikací může probíhat.

3.1 Programovací jazyky mobilních zařízení

S faktem, že mobilní zařízení zastupují na trhu s elektronikou v dnešní době jedno z hlavních (ne-li nejdůležitější) odvětví, je pro vývoj mobilní aplikace zásadní řádná analýza. Tím je myšleno, že pro vytvoření úspěšné aplikace je důležité nejprve podniknout kroky, zahrnující průzkum trhu, naplánování strategie vývoje a také uvědomění si dostupného rozpočtu. Strategie vývoje v sobě pak zahrnuje výběr programovacího jazyka, pod kterým bude aplikace vytvořena. Před výběrem samotného jazyka je nicméně důležité položit si několik zásadních otázek:

- *Které programovací jazyky jsou v současnosti "in" k vývoji aplikací pro mobilní platformy?*
- *Jaký jazyk využít, aby aplikace fungovala efektivně a využívala co nejméně prostředků koncového zařízení?*
- *Jakým způsobem lze konkrétní jazyk využít s ohledem na schopnosti a zkušenosti vývojáře?*

Po zodpovězení si na tyto otázky je následně jednodušší vybrat si z následujících nejpopulárnějších programovacích jazyků pro vývoj mobilních aplikací.

3.1.1 Java

Java patří jednoznačně mezi nejoblíbenější programovací jazyk pro tvorbu mobilních aplikací zejména pro OS *Android* a *iOS*. Nabízí hlavně jednoduchou přenositelnost a opakovanou použitelnost kódu, který lze spustit v různých prostředích, virtuálních strojích, prohlížečích a napříč platformami. Má také velkou podporu open source s mnoha knihovnami a nástroji, které usnadňují vývojářům tohoto jazyka práci.

3.1.2 JavaScript

Pro vývoj aplikací se ve velké míře využívá i tento široce uznávaný jazyk. Mezi jeho výhody patří nezávislost na platformě, tzn. i tento jazyk je snadno přenositelný a použitelný. Jeho použití se uplatňuje kromě vývoje mobilních aplikací také k prohlížení webu. Co se týče jeho standardizace, neexistuje žádný zavedený standard, což vede k velkému variačnímu rozpětí stylu psaní zdrojového kódu tohoto jazyka.

3.1.3 PHP

Tento jazyk je obecně doporučován vývojářům, kteří pracují s databázemi. Je efektivně využíván pro používání skriptů, příkazového řádku i vývojem aplikací. Často se o něm tvrdí, že jako open source je výjimečně snadno integrovatelný a lehký k naučení.

3.1.4 Python

S programovacím jazykem *Python* se lze například setkat ve webových aplikacích, desktopech, síťových serverech nebo mediálních nástrojích. Sází na svou jednoduchost, jednoznačnost a úspornost,

díky čemuž je snadno čitelný. Jedná se o přívětivý jazyk pro osoby, snažící naučit se programovat, neboť je doporučován pro svou jednoduchost k pochopení složitostí vývoje aplikací obecně. Nachází uplatnění i u vývoje aplikací pro OS *Android* a *iOS*.

3.1.5 **Swift**

Swift je programovací jazyk speciálně navržený společností *Apple* pro práci s platformami *iOS* a *OS X*. Jako open source se jednoduše učí zejména vývojářům, kteří mají zkušenosti s programovacím jazykem *Objective-C*.

3.1.6 **Kotlin**

Nejnovějším přírůstkem na poli vývoje mobilních aplikací je programovací jazyk *Kotlin*. Jedná se o moderní jazyk, kde je kladen větší důraz na zredukování obsahu zdrojového kódu, čímž má vývojář usnadněnou práci s jeho následným testováním a údržbou. Velkou výhodou je interoperabilita tohoto programovacího jazyka s *Javou*. Zdrojový kód založený na jazyce Java lze tak volat pomocí kódu Kotlinu nebo naopak. V současné době stále více vývojářů mobilních aplikací začíná preferovat právě tento jazyk před jazykem *Java*, nicméně to rozhodně neznamená, že by využití jazyka *Java* mělo zanikat.

[32]

3.2 **Vývojové nástroje**

Jakmile je vývojář rozhodnut, se kterým programovacím jazykem bude pracovat, přechází do fáze výběru vývojového nástroje. Jedná se o software navržený tak, aby pomáhal s tvorbou mobilních aplikací. Toho lze dosáhnout několika způsoby jelikož existují jednak nativní mobilní vývojové nástroje, ale také mobilní vývojové nástroje multiplatformní (tzv. *cross-platform*).

- **Nativní vývojové nástroje** – Pomohou s vývojem specializovaných aplikací, které mohou využít všech funkcí dané platformy. Pak ale ovšem také záleží na již dříve zmíněné a vysvětlené podpoře API. Nativní aplikace jsou tedy vyvíjeny pro konkrétní mobilní platformy v jejich rodném programovacím jazyce. Jedná se opět o dvojici *Android* a *iOS*, kde první jmenovaná platforma využívá jazyků *Java* a *Kotlin* a druhá *Objective-C* a *Swift*. V nedávné době bychom zde mohli uvést dnes již nepopulární platformu *Windows Phone* s jazykem *C#,* pro kterou oficiálně skončila podpora na konci roku 2019.
- **Multiplatformní vývojové nástroje** – Tento typ vývojových nástrojů umožňuje vytvořit aplikace, které nejsou konstruovány pro jednu konkrétní platformu, ale pro více současně, což snižuje náklady a čas, potřebný k jejich vývoji. S těmito výhodami ovšem přichází i důsledky, za které lze považovat nižší kvalita softwaru a více chyb či bugů ve srovnáním s nativními vývojovými nástroji.

Na internetu existuje nespočet vývojových prostředí – ať už nativních nebo multiplatformních. Nyní je zde nutné zmínit vývojové prostředí *Android Studio*, neboť právě toto nativní prostředí je ideální volbou pro vývoj mobilní aplikace, která je předmětem této práce.

[33]

3.3 Android Studio

Jedná se o oficiální *integrované vývojové prostředí* (IDE) pro vývoj aplikací s OS *Android*. Díky ergonomickému designu a inteligentní asistenci ve formě našeptávání je vývoj aplikací v tomto prostředí nejen atraktivní, ale nabízí i množství funkcí, zvyšujících produktivitu vývojářů. Jedná se například o:

- Rychlý emulátor s mnoha funkcemi.
- Flexibilní systém sestavování aplikací *Gradle*.
- Aplikování změn zdrojového kódu a zdrojů bez nutnosti restartování/ukončování aplikace.
- Integrace platformy *GitHub*, import kódu.
- Rozsáhlé testovací nástroje jako monitoring výkonu, kompatibilita, atd.
- Podpora C++ a NDK (*Native Development Kit* – nástroj, umožňující použití kódu C/C++ včetně knihoven).

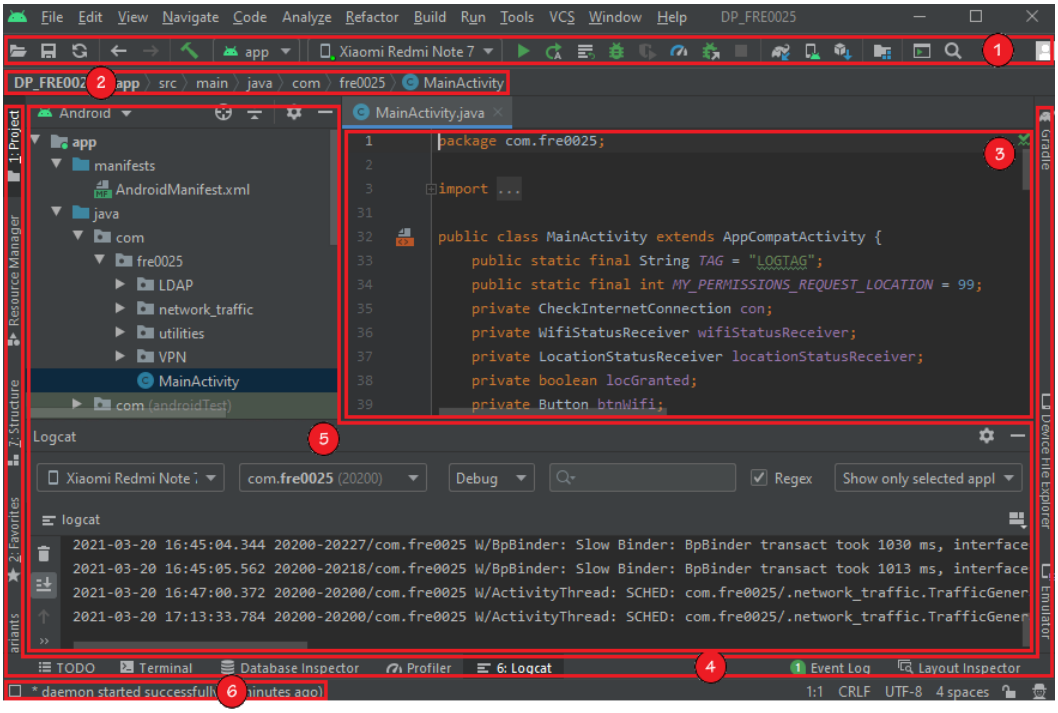
Projekty se v *Android Studiu* skládají z jednoho nebo více modulů. Jedná se o kolekce souborů nebo sestavení, které rozdělují projekt na jednotlivé funkční jednotky.

Soubory se zdrojovým *Java* kódem náleží do modulu *java*. Mezi zdroje, přidružené k aplikaci, lze uvést ku příkladu bitmapy nebo soubory s příponou XML, které popisují rozložení grafických prvků aplikace nebo řetězce. Tyto zdroje jsou dále organizovány do odpovídajících modulů nižší úrovně a vycházejí z kořenového modulu zvaného *res*.

[34]

3.3.1 Uživatelské prostředí

Hlavní okno Android Studia je uspořádáno na několik logických celků, což lze vypozorovat z obrázku 3.1. Většina oken či prvků GUI se dá rozbalit, sbalit nebo případně posunout pro omezení rušivých elementů při programování a lepší organizaci pracovního prostoru. Android Studio rovněž podporuje velkou škálu klávesových zkratek, aby byla práce v jeho prostředí co nejefektivnější.



Obrázek 3.1: *Android Studio - rozložení uživatelského prostředí*

1. **Panel nástrojů** – Poskytuje spouštěcí nástroje aplikace a případně nástroje instalovaných doplňků.
2. **Navigační lišta** – Umožňuje kompaktní pohled na projekt, kde definuje cestu k právě otevřenému souboru.
3. **Editor** – Slouží k tvorbě a úpravě zdrojového kódu.
4. **Panel oken** – Umožňuje rozbalení nebo sbalení jednotlivých oken.
5. **Specifická okna** – Poskytují přístup k úkolům, terminálu projektu a monitorují historii verzí projektu, jeho sestavení a v neposlední řadě také *Logcat*, což je nástroj příkazového řádku, vypisující zprávy ať už se jedná o chyby, varování nebo informační hlášení.
6. **Stavový řádek** – Zobrazuje aktuální stav projektu.

[34]

3.3.2 Soubor AndroidManifest.xml

Každá aplikace dále obsahuje modul *manifests* se souborem *AndroidManifest.xml*. Tento soubor musí být součástí každého projektu a poskytuje o aplikaci základní informace, které musí znát překladač, OS *Android* koncového zařízení a v případě umístění aplikace na *Google Play* i tato platforma. V tomto souboru je mimo jiné definován název balíčku aplikace, který překladač používá jednak pro získání lokací zdrojových kódů a zdrojových souborů při budování projektu a také jako název jmenného prostoru pro vygenerování speciální třídy *R.class*, která zajišťuje přístup k těmto zdrojovým souborům. Název balíčku v souboru *AndroidManifest.xml* by se měl tedy vždy shodovat s názvem balíčku samotného projektu. V případě dalších dílčích balíčků v projektu je nicméně nutné, aby jejich soubory měly importovanou třídu *R.class* právě pomocí názvu balíčku z již dříve zmíněného XML souboru. Název balíčku je pak při kompilaci výsledného APK souboru nahrazen identifikátorem získaným ze systému *Gradle*, a to pro jednoznačnou identifikaci aplikace na mobilním zařízení či eventuelně na *Google Play*.

Další nutnou položkou souboru *AndroidManifest.xml* je uvedení komponent, ze kterých se aplikace skládá. Mezi ně se řadí zejména následující: aktivita, služba, „naslouchač“ oznámení (tzv. *broadcast receiver*) a poskytovatel obsahu, zmíněný v kapitole 2.5.5. Při definování dílčích balíčků je princip tečkové notace stejný, nýbrž za úvodní tečkou je nutné zmínit jejich odpovídající názvy. Volání všech komponent je aktivováno pomocí tzv. *intent*, což je zpráva, která žádá o akci z jiné součásti aplikace. Většinou se objekt *Intent* využívá při startu aktivity, služby nebo při poskytování broadcast zpráv. Když aplikace řeší *intent*, systém lokalizuje komponentu, která jej dokáže zpracovat na základě položky v *intent-filteru* a následně spustí její odpovídající instanci a předá ji odpovídající objekt. *Intent-filter* je součástí manifestu.

Oprávnění jsou také položkou manifest souboru a jsou vyžadována, pokud aplikace potřebuje přístup k určitým systémovým funkcím mobilního zařízení (přístup k internetu, souborovému systému, fotoaparátu, GPS atd.) nebo k uživatelským datům jako jsou kontakty, SMS, kalendář atd. Od verze *Android 6.0* lze řešit povolení či zamítnutí těchto oprávnění již při běhu aplikace, kdy uživatel dostane při jejím prvním spuštění na výběr, zda je chce aplikaci udělit či nikoliv. Pokud uživatel oprávnění aplikaci neudělí, je přístup k dané systémové funkci, respektive citlivým datům zakázán. Tento stav je potom možné změnit pouze manuální změnou oprávnění u požadované aplikace v nastavení mobilního zařízení nebo případnou reinstalací aplikace a následným povolením konkrétního oprávnění při opětovném prvním spuštění.

V manifestu je možné dále deklarovat kompatibilitu hardwarových a softwarových funkcí, a tím tedy vymezit konkrétní skupinu mobilních zařízení, pro které je aplikace určena. Funkčnost aplikace je potom závislá na tom, zda mobilní zařízení splňuje daná kritéria kompatibility.

[34][35]

3.3.3 **Nástroj Gradle**

Integrovaný nástroj pro sestavování projektů, zvaný *Gradle*, funguje v prostředí Android Studia nezávisle na příkazovém řádku a slouží jako zprostředkovatel následujících funkcí:

- uživatelské přizpůsobení procesu sestavování aplikací,
- tvorba více APK souborů pro konkrétní projekt – Uplatnění zejména při vyvíjení aplikace pro různé varianty rozlišení obrazovky cílových mobilních zařízení nebo při vývoji bezplatné vs. placené verze aplikace,
- správa závislostí (tzv. *dependencies*).

Nástroj Gradle tedy umožňuje zprostředkovat výše zmíněné funkce, aniž by se muselo zasahovat do souborů se zdrojovými kódy projektu. Namísto toho tento nástroj využívá sestavovací soubory, zvané *build.gradle*.

Každý projekt má jeden sestavovací soubor nejvyšší úrovně a další soubory sestavení na úrovni modulů, které jsou automaticky generovány při importu existujícího projektu do prostředí Android Studia. Za zmínku pak stojí také nutnost uvést v souboru *build.gradle* minimální podporovanou SDK sadu aplikace, o kterých již byla zmínka v kapitole 2.1.

[34]

4 Vývoj aplikace pro generování síťového provozu

Tato kapitola pojednává o vývoji aplikace, která generuje síťový provoz. Jednotlivé části aplikace, mezi které lze zahrnout podporu pro platformu *OpenVPN* nebo přístup k adresářovému serveru, nejprve popisuje z teoretického hlediska. Dále jsou zde charakterizovány internetové protokoly HTTP, FTP, IMAP, POP3, SIP a SMTP, jelikož síťový provoz je v aplikaci generován právě v jejich režii.

Podrobná dokumentace vývoje je rovněž součástí této kapitoly. Implementace jednotlivých komponent aplikace ve vývojovém prostředí *Android Studio* vždy následuje po teoretickém úvodu, který souvisí s danou problematikou.

Následující kapitola nastiňuje účel vývoje samotné aplikace, respektive jak se může aplikace uplatnit v praxi. Další oddíl pak objasňuje princip fungování této aplikace, jehož součástí je rovněž vývojový diagram v grafické podobě.

4.1 Uplatnění v praxi

Název aplikace odkazuje na generování síťového provozu. Pod tímto pojmenováním si však člověk může představit více scénářů, kterých by se dalo pod tímto označením prakticky realizovat. Proto je nyní na místě detailně objasnit, k jakým účelům tato aplikace vlastně slouží a kde konkrétně by se dala uplatnit.

Aplikace najde uplatnění v oblasti bezpečnosti bezdrátových sítí Wi-Fi. Slouží totiž de facto jako tzv. *honeypot*, což je pojem z oblasti informatiky definující systém, který má za úkol nalákat potencionálního útočníka na citlivá data uživatelů a odhalit tak jeho totožnost. Aplikace je navržena tak, že záměrně využívá nezabezpečeného přenosu citlivých dat bezdrátovou sítí, která jsou tím pádem snadno dohledatelná a zjistitelná při použití vhodného softwaru pro analýzu síťového provozu.

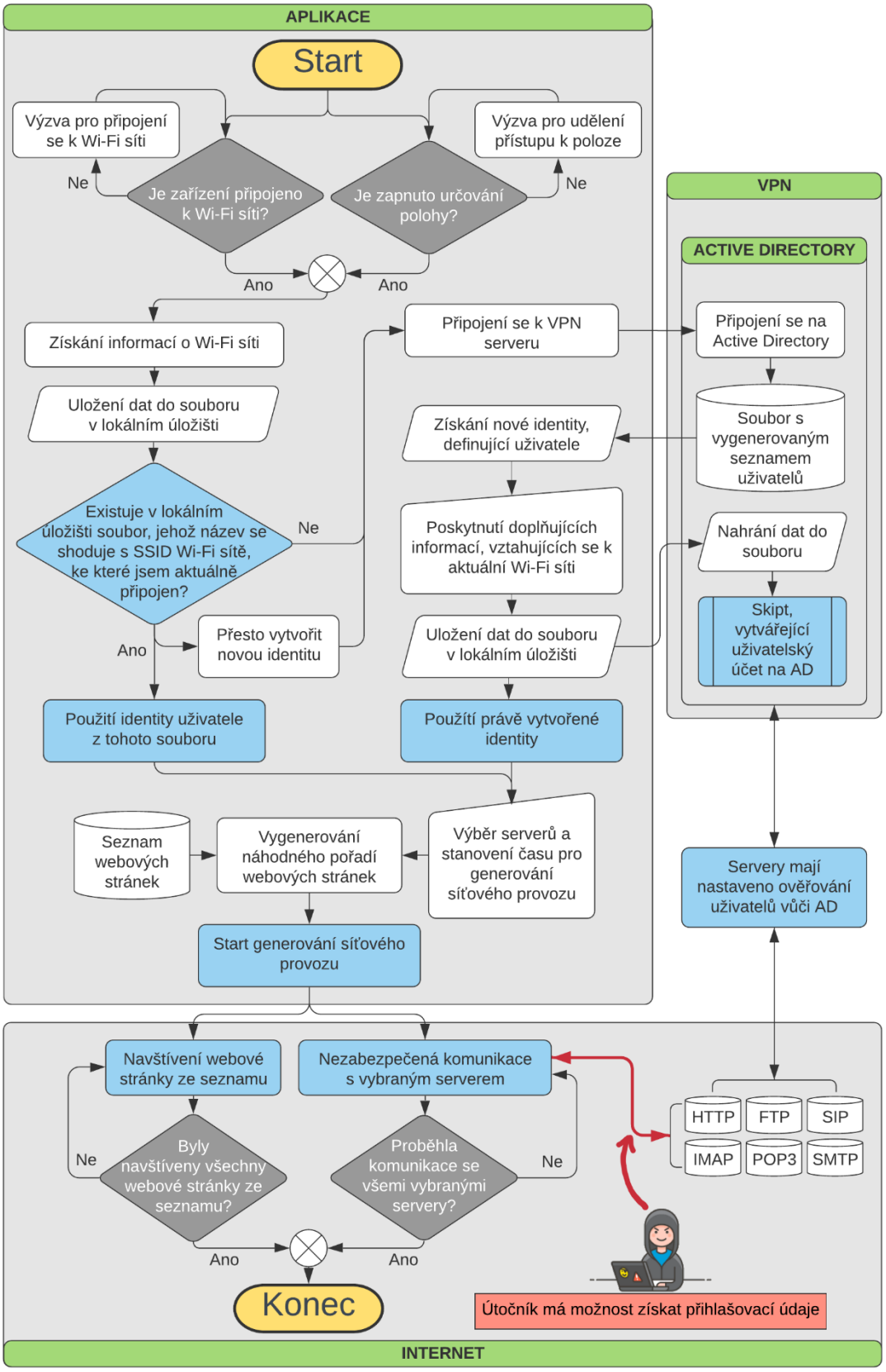
Použít by se mohla zejména ve veřejně přístupných Wi-Fi sítích, jejichž implementace je v dnešní době široce rozšířená zejména ve školách, obchodních centrech, restauracích, kavárnách atp.

4.2 Princip fungování

Generovat síťový provoz v této aplikaci znamená snahu nahradit chování typického uživatele internetu jako celku. V praxi to tedy vypadá tak, že aplikace navštěvuje v různých logických intervalech webové servery, které běžný uživatel internetu často využívá v reálném světě. Pro věrohodnější napodobení chování běžného uživatele může aplikace s webovými servery provádět další interakce – například otevření článků na zpravodajském webu, přihlášení se na e-mailový online portál, stažení souboru z FTP serveru atd. Pokud by tento síťový provoz monitoroval správce sítě nebo útočník, viděl by v drtivé většině případů komunikaci jako šifrovanou, neboť webové servery využívají síťových protokolů SSL/TLS.

Hlavním důvodem generování síťového provozu je však v této práci nešifrovaná komunikace, pro kterou je z hlediska aplikace jednak důležité napojení na LDAP server (konkrétně *Active Directory* – dále jen AD) a jednak konfigurace odpovídajících serverů, se kterými může aplikace nešifrovaně komunikovat.

K pochopení funkčnosti celé aplikace, respektive celého projektu, slouží následující vývojový diagram. Modrou barvou jsou na něm zvýrazněny klíčové prvky.



Obrázek 4.1: *Vývojový diagram aplikace*

Adresářový server (AD) totiž aplikaci poskytuje potřebná data (přihlašovací údaje), která pak aplikace nešifrovaně distribuuje po síti prostřednictvím protokolů ze zadání této práce. Tyto přihlašovací údaje se na AD vyskytují v konkrétním souboru, s nímž aplikace pracuje. Nejprve je však nutné realizovat přístup k samotnému AD, k čemuž aplikace využívá připojení prostřednictvím VPN. K souboru, který pak obsahuje přihlašovací údaje, aplikace přistupuje s využitím síťového komunikačního protokolu SMB, jež je součástí aplikační vrstvy modelu OSI.

Tento soubor obsahuje dlouhý seznam, kde každému jeho řádku odpovídá právě jeden uživatel, respektive jeho přihlašovací údaje. Z tohoto seznamu si aplikace pokaždé vyjme jednoho uživatele, ke kterému následně přidá informace o Wi-Fi síti, k níž je momentálně připojen, a uloží je do specifického souboru zpět na server. Na serveru je spuštěn *PowerShell* skript, který hlídá změny na souborovém systému pro adresář, ve kterém se tento soubor nachází. Při každém uložení souboru do tohoto adresáře skript analyzuje jeho obsah, čímž získá o uživateli a přidružené bezdrátové síti požadované informace. Pomocí těchto informací skript vytvoří na AD nového uživatele.

Každý takto vytvořený uživatel má tedy na AD svůj záznam, pod jehož identitou následně aplikace v síti vystupuje. Na adresářovém serveru se vyskytuje položka *sAMAccountName*, která definuje uživateli přihlašovací jméno. Společně s tímto jménem má každý uživatel své unikátní heslo.

Konfigurace serverů, se kterými aplikace nešifrovaně komunikuje, byla rovněž předmětem této práce. Na serverech je nainstalován následující software, odvíjející se od protokolů, zmíněných v zadání této práce:

- *Apache2* – HTTP
- *Proftpd* – FTP
- *Dovecot* – IMAP, POP3
- *Postfix* – SMTP
- *Asterisk* – SIP

Pozn.: (software – protokol)

Generování síťového provozu pro tyto protokoly je v aplikaci řešeno implementací odpovídajících klientů, kteří s nakonfigurovanými servery komunikují. Identita uživatele, kterou musí jednotliví klienti aplikace poskytnout serverům v nešifrované podobě, pak odpovídá charakteristice jednotlivých protokolů. V případě HTTP protokolu se jedná o jednoduché vložení přihlašovacích údajů do předem připraveného formuláře na webových stránkách serveru. Dále aplikace nahrává soubory na FTP server a ownCloud server, kdy pro přístup k těmto serverům je rovněž nutné poskytnout přihlašovací údaje. V případě e-mailového klienta je zase potřeba poskytnout přihlašovací údaje pro přístup k e-mailovému serveru, na němž lze dále generovat síťový provoz stahováním pošty do lokálního úložiště nebo posláním e-mailu. Poskytnutí přihlašovacích údajů v rámci SIP protokolu je zase nutné využít při registraci uživatele na SIP server.

Vývoj aplikace pro generování síťového provozu je součástí komplexnějšího projektu, který má výše zmíněné servery pod svou správou, čímž má možnost analyzovat a hlídat stav, kdy se uživatel s konkrétními přihlašovacími údaji na nezabezpečený server přihlásí. Pokud se tak stane, pomocí záznamů v AD lze snadno dohledat, zda se na určité bezdrátové síti vyskytl útočník, neboť se pod údaji, které byly Wi-Fi sítí přenášeny v rámci generování síťového provozu, na servery později přihlásil sám.

Tímto způsobem se tedy z dlouhodobého hlediska naskýtá jistá možnost monitorovat bezpečnost bezdrátových sítí Wi-Fi a schopnost odhalit případné odposlouchávání provozu.

4.3 Úvodní obrazovka aplikace

Úvodní stránka definuje vstupní bod aplikace. Uživateli je zde stručně vysvětleno, k čemu aplikace slouží a řeší se zde udělení potřebných oprávnění, bez kterých by aplikace nemohla fungovat. Konkrétně se jedná o povolení pro přístup k poloze zařízení, neboť bez této vlastnosti není možné od verze Android 8.0 zjišťovat název Wi-Fi sítě, ke které je zařízení připojeno, což je pro tuto aplikaci později vyžadováno.

Další částí úvodní aktivity aplikace je jednoduché ověření, které kontroluje následující:

- Je zařízení připojeno k Wi-Fi síti, která má přístup k internetu?
- Je v zařízení zapnutý přístup k poloze?

```
public class MainActivity extends AppCompatActivity {
    private CheckInternetConnection con;
    private WifiStatusReceiver wifiR;
    private LocationStatusReceiver locR;
    private boolean locGranted;
    ...
    @Override
    protected void onStart() {
        super.onStart();
        registerReceiver(wifiR,
                new IntentFilter(WifiManager.WIFI_STATE_CHANGED_ACTION));
        registerReceiver(locR,
                new IntentFilter(LocationManager.PROVIDERS_CHANGED_ACTION));
        registerReceiver(wifiListener, new IntentFilter("Wifi"));
        registerReceiver(locListener, new IntentFilter("Loc"));
    }
```

Ke zjištění stavu Wi-Fi sítě a polohy slouží třídy *WifiStatusReceiver* a *LocationStatusReceiver*. Jedná se o *broadcast receiver* třídy, jejichž princip odpovídá návrhovému vzoru *publish-subscribe* a volají se v případech, kdy aplikace potřebuje dynamicky reagovat na konkrétní události, pro které se registrují. V tomto případě se jedná o změny stavu proměnných *WIFI_STATE_CHANGED_ACTION* a *PROVIDER_CHANGED_ACTION*, které jsou součástí systému. Jedná se o parametry objektů *IntentFilter*, které popisují, o jaké objekty se přidružená komponenta zajímá. *Intenty* obecně zprostředkovávají komunikaci mezi jednotlivými komponentami aplikace nebo systémovými službami. Pokaždé, když se tedy v zařízení změní status Wi-Fi sítě nebo polohy, zavolá se odpovídající třída, ve které se dále řeší, zda došlo k zapnutí nebo vypnutí dané služby.

```
    BroadcastReceiver wifiListener = new BroadcastReceiver() {
        @Override
        public void onReceive(Context c, Intent i) {
            boolean enabledWifi = i.getExtras().getBoolean("enabledWifi");
            if(enabledWifi) {
                ProgressBarWorker pbw = new ProgressBarWorker();
                pbw.startProgressDialog(c);
                Handler handler = new Handler();
                handler.postDelayed(() -> {
                    if(con.checkWifiStatusAndConnection(c))
                        pbw.stopProgressDialog();
                    wifiStatusReceiver.shouldStartProgressBar(false);
                    checkBtnNext();
                }, 10000);
```

```
            } else {
                wifiStatusReceiver.shouldStartProgressBar(true);
                checkBtnNext();
            }
        }
    };

    BroadcastReceiver locListener = new BroadcastReceiver() {
        @Override
        public void onReceive(Context c, Intent in) {
            checkBtnNext();
        }
    };
```

Po zjištění stavu Wi-Fi sítě nebo polohy je výstup tříd *WifiStatusReceiver* a *LocationStatusReceiver* stejným způsobem zpracováván i v aktivitě, která na tento příchozí stav reaguje lokálními třídami *wifiListener* a *locListener*.

```
    private void checkBtnNext() {
        if(wifiStatus() && locationStatus()) {
            btnNext.setVisibility(View.VISIBLE);
            btnNext.setEnabled(true);
        } else {
            btnNext.setVisibility(View.INVISIBLE);
            btnNext.setEnabled(false);
        }
    }

    private boolean wifiStatus() {
       boolean x = con.checkWifiStatusAndConnection(getApplicationContext());
        wifiR.shouldStartProgressBar(!x);
        return x;
    }

    private boolean locationStatus() {
        LocationManager lm = (LocationManager)getApplicationContext()
                                .getSystemService(Context.LOCATION_SERVICE);
        return lm.isProviderEnabled(LocationManager.GPS_PROVIDER);
    }
```

Metoda *checkBtnNext* nakonec kontroluje, za jsou obě služby zapnuty a pokud ano, uživateli se zobrazí tlačítko, umožňující přechod do další části aplikace.

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        btnNext.setOnClickListener(view -> {
            if(locGranted) {
                new HandlingWifiInfo()
                                .saveWifiInfoToFile(getApplicationContext());
                Intent i = new Intent(MainActivity.this, InfoActivity.class);
                startActivity(i);
            }
        });
    }
} //class MainActivity
```

Po kliknutí na tlačítko se rovněž zavolá metoda *saveWifiInfoToFile*, jejímž úkolem je shromáždit požadované informace o Wi-Fi síti, ke které je zařízení momentálně připojeno. S těmito informacemi potom aplikace dále pracuje. Grafická podoba úvodní obrazovky aplikace je zřejmá z obrázku 4.2.

Obrázek 4.2: *Úvodní obrazovka aplikace*

4.4 Virtuální privátní sítě

Jelikož je v této práci nutné komunikovat se vzdáleným adresářovým serverem, přichází na řadu kapitola o virtuálních privátních sítích, které umožňují přístup do vzdálených lokálních sítí.

Pod pojmem virtuální privátní síť (VPN) si můžeme v současnosti představit velmi rozšířenou síťovou technologii, jejímiž primárními cíli jsou:

- **Poskytnout uživateli online soukromí a anonymitu vytvořením zabezpečeného a šifrovaného připojení (tunelu) přes veřejnou síť.**

VPN poskytuje uživateli anonymitu v internetovém prostředí tím, že ukrývá jeho síťovou aktivitu pod IP adresou samotného VPN serveru. To znamená, že se uživatel stává nevystopovatelným jednak vůči hackerům, korporacím a státní správě, ale i vůči samotnému poskytovateli internetového připojení, který tak rovněž nemůže aktivitu vystopovat, a ví pouze to, že uživatel komunikuje se vzdáleným VPN serverem. Při použití virtuální privátní sítě tedy nebude nikdo (kromě samotného VPN poskytovatele) schopen o uživateli zjistit informace o IP adrese s jejím umístěním nebo historii prohlížení. Další výhodou používání VPN je schopnost podvrhnout svou geografickou polohu, neboť pokud je poskytovatel VPN dostatečně robustní, nabízí zpravidla přístup ke svým VPN serverům, které se nacházejí v různých zemích. Díky této schopnosti lze získat přístup k webovým stránkám, které jsou pro některé země či oblasti blokovány.

Při výběru VPN služeb je nicméně důležité dbát na to, aby byl jejich poskytovatel bezpečný a důvěryhodný. V neposlední řadě je nutné věnovat pozornost také tomu, zda není poskytovaná IP adresa VPN serveru sdílena mezi více zákazníků, čímž se může pro některé internetové stránky dostat

na černou listinu. V porovnání s obecnými výhodami VPN připojení jsou však tyto záležitosti docela zanedbatelné.

- **Umožnit přístup do vzdálené lokální sítě (např. soukromé, podnikové a školní).**

Kromě anonymního používání internetu však byla tato technologie původně vyvinuta zejména pro přístup zaměstnanců k firemní síti z domova. Tento nápad se záhy široce uplatnil i mezi běžnými spotřebiteli. Na základě této myšlenky existují dva základní typy VPN připojení. První nese název *vzdálený přístup*, kdy se uživatelé připojují do určité lokální sítě vzdáleně z různých míst. Druhým typem je pak tzv. *site-to-site VPN*, která se využívá především v rozlehlých podnikových prostředích, například mezi pobočkami (intranet).

Mezi způsoby zabezpečení, které virtuální privátní sítě využívají, patří protokoly PPTP, IPSec, L2TP, SSL, TLS, SSH.

Implementace podpory pro VPN je v této práci realizována prostřednictvím open source softwaru *OpenVPN*, který patří mezi nejrozšířenější VPN řešení vůbec. *OpenVPN* funguje na principu klient-server, kdy k jednomu VPN serveru může přistupovat více klientů. Pro šifrování a autentizaci zde slouží knihovna *OpenSSL* a pro přenos dat lze využít jak TCP, tak UDP protokol.

Jelikož je potřeba se k VPN serveru připojit, je zřejmé, že zařízení, na kterém je vyvíjená aplikace nainstalována, bude vystupovat jako VPN klient. Abychom se jako klient mohli k serveru připojit, musí nám poskytnout požadované informace, jak připojení realizovat. Samotná konfigurace VPN serveru v sobě zahrnuje také způsoby, jak se k němu mohou klienti připojovat. Děje se tak pomocí konfiguračního souboru, který musí server klientům poskytnout. Jedná se o textový soubor s příponou *ovpn*, který klientům specifikuje jednotlivé parametry VPN serveru – použitý transportní protokol, IP adresu serveru a port, možnosti opětovného připojování při slabé konektivitě atd. Dalšími důležitými parametry jsou sdílený klíč, digitální certifikát a certifikační autorita. Ty se mohou vyskytovat i mimo konfigurační soubor jako samostatné soubory (zpravidla s příponou *pem*). V tom případě je pak nutné uvést do konfiguračního souboru cesty k jejich umístění. Obsahem těchto tří souborů jsou již data v zašifrované podobě, která slouží k autentizaci klienta, respektive řeší, zda má uživatel (klient) oprávnění se k danému serveru připojit.

[37][38]

4.4.1 Implementace VPN do aplikace

Připojení k VPN serveru je v aplikaci řešeno implementací open source klientského softwaru *ics-openvpn*, jehož repositář je volně dostupný na platformě *GitHub*. Tento software je založen na komunitní verzi oficiálního *OpenVPN* řešení pro OS Android – aplikaci *OpenVPN Connect*.

Začlenění tohoto softwaru do vyvíjené aplikace lze v Android Studiu realizovat naklonováním Git repositáře do vlastního projektu. Po úspěšném naklonování je v projektu vytvořen nový modul, odpovídající tomuto VPN řešení. Společně s ním je sestavovacím nástrojem *Gradle* vytvořen nový soubor *build.gradle*, který odpovídá speciálně tomuto modulu. VPN řešení je tímto v projektu dostupné a pro jeho využití v aplikaci zbývá jen zavolat odpovídající metody v odpovídajících třídách. Třída, která implementuje základní funkčnost celého VPN modulu se nazývá *VPNFragment*.

```
import de.blinkt.openvpn.OpenVpnApi;
import de.blinkt.openvpn.core.OpenVPNService;
import de.blinkt.openvpn.core.OpenVPNThread;

public class VPNFragment extends Fragment implements View.OnClickListener,
ChangeVPNServer, GetVpnStatus {
    private VPNServer server;
    private SharedPreference preference;

    ...

    @Override
    public View onCreateView(LayoutInflater i, ViewGroup v, Bundle b) {
        binding = DataBindingUtil.inflate(i, R.layout.fragment_vpn, v, false);
        View view = binding.getRoot();
        LocalBroadcastManager.getInstance(getActivity()).registerReceiver(
                    broadcastReceiver, new IntentFilter("connectionState"));

        preference = new SharedPreference(getContext());
        server = preference.getServer();
        return view;
    }

    @Override
    public void onViewCreated(View view, Bundle b) {
        super.onViewCreated(view, b);
        binding.vpnBtn.setOnClickListener(this);
        binding.vpnBtnDisconnect.setOnClickListener(this);
        binding.vpnBtnNextActivity.setOnClickListener(this);
        setStatus(OpenVPNService.getStatus());
    }
```

Z modulu *ics-openvpn* stačí naimportovat třídy *OpenVpnApi*, *OpenVPNService* a *OpenVPNThread*, které v aplikaci postačují k realizaci vlastního VPN připojení. Třída *VPNFragment* je potomkem třídy *Fragment*, díky čemuž je implementováno chování veřejných metod *onCreateView* a *onViewCreated*. V prvně jmenované metodě se vytváří instance *XML layout* souboru s rozložením do odpovídajícího objektu *View* a také se zde registruje lokální *broadcastReceiver*, jehož úkolem je po úspěšném připojení se vypisovat aktuální parametry připojení (čas, download a upload). Druhá metoda obsahuje posluchače událostí na kliknutí pro 3 tlačítka, nacházející se ve fragmentu. Parametry *this* se zde odkazují na metodu *onClick*, která tlačítkům implementuje odpovídající chování. V metodě *onViewCreated* se rovněž nastavuje výchozí status VPN připojení, který zjistí importovaná třída VPN modulu *OpenVPNService*.

```
    @Override
    public void onClick(View v) {
        switch (v.getId()) {

            case R.id.vpnBtn: {
                startVpn();
                break;
            }
            case R.id.vpnBtnDisconnect: {
                if(vpnStart) {
                   AlertDialog.Builder b = new AlertDialog.Builder(getActivity());
                   b.setMessage(getActivity().getString(R.string.vpnClose));
                   b.setPositiveButton(getActivity().getString(R.string.yes),
                                       (dialog, id) -> stopVpn());

                   b.setNegativeButton(getActivity().getString(R.string.no),
                                       (dialog, id) -> {});
```

```
                    AlertDialog dialog = b.create();
                    dialog.show();
                }
                break;
            }
            case R.id.vpnBtnNextActivity: {
                Intent openAct = new Intent(getActivity(), SmbActivity.class);
                startActivity(openAct);
            }
        }
    }
```

Zmíněná 3 tlačítka slouží k následujícím účelům: připojení se k VPN, odpojení se a přechod na další aktivitu aplikace. V aplikaci je v počátečním stavu zobrazeno pouze tlačítko k připojení se. Pokud se aplikace úspěšně k VPN připojí, tlačítko pro připojení přejde do stavu *INVISIBLE* a nahradí ho zbylá 2 tlačítka (odpojení se a přechod do další části aplikace). Kliknutím na tlačítko pro odpojení se se vyvolá dialog, ve kterém je uživatel dotázán, zda se chce opravdu odpojit. Odpojení pak řeší třída *OpenVPNThread* metodou *stop*. Stav všech tří tlačítek se při odpojení vrátí do počátečního stavu. Pokud se aplikace k VPN úspěšně připojila, kliknutím na tlačítko *vpnBtnNextActivity* může uživatel přejít do další části aplikace.

Proces připojení se k VPN serveru již není tak triviální. Jeho algoritmus musí projít určitými stavy, mezi které patří v rámci této aplikace zejména autentizace. Ta se v případě *OpenVPN* řeší pomocí konfiguračních souborů s příponou *ovpn*, jak již bylo v této kapitole uvedeno dříve.

```
    private void startVpn() {
        try {
            InputStream is = getActivity().getAssets().open(server.getOvpn());
            InputStreamReader isr = new InputStreamReader(is);
            BufferedReader br = new BufferedReader(isr);
            StringBuilder sb = new StringBuilder();
            String line;

            while (true) {
                line = br.readLine();
                if (line == null) break;
                sb.append(line).append("\n");
            }

            br.readLine();
            OpenVpnApi.startVpn(getContext(),
                                sb.toString(),
                                server.getCountry(),
                                server.getOvpnUserName(),
                                server.getOvpnUserPassword());

            binding.vpnBtn.setText(R.string.connecting);
            vpnStart = true;
        } catch (IOException | RemoteException e) {
            e.printStackTrace();
        }
    }
} //class VPNFragment
```

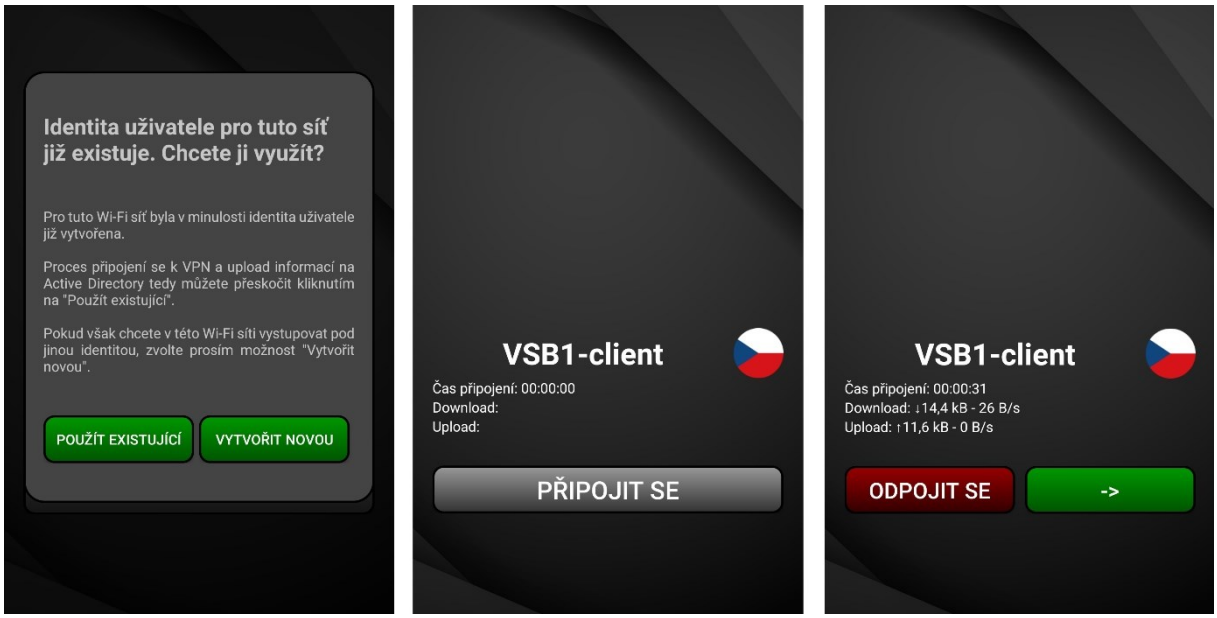
Konfigurační soubor VPN serveru se v projektu *Android Studia* nachází ve složce *assets*, do kterých se běžně ukládají soubory, jejichž nezpracovaná data je potřeba číst. Obsah tohoto souboru se musí načíst do lokální proměnné, aby se mohl následně předat jako parametr datového typu *String* metodě *startVpn*, která je definována ve třídě *OpenVpnApi*. Proměnná *server* zde představuje instanci

třídy *VPNServer*, která slouží pouze k seskupení jednotlivých parametrů serveru. Mezi tyto parametry patří vlastní název VPN serveru, název státu, URL obrázku odpovídající státní vlajky a název konfiguračního souboru. Tyto parametry vycházejí z implementace modulu *ics-openvpn*. Metoda *startVpn* rovněž obsahuje parametry *getOvpnUserName* a *getOvpnUserPassword*, což jsou další metody třídy *VPNServer*. Některé VPN servery totiž požadují autentizaci pomocí jména a hesla. V tomto případě jsou hodnoty těchto dvou parametrů *null*, neboť využívaný VPN server je nevyžaduje, nicméně je potřeba je uvést.

Výše zmíněná třída *VPNFragment* spadá pod aktivitu s názvem *VPNActivity*. Tato aktivita mimo jiné kontroluje, zda v externím úložišti existuje soubor, jehož název se shoduje s názvem Wi-Fi sítě, ke které je zařízení momentálně připojeno. Ke zjištění aktuálního názvu Wi-Fi sítě slouží pomocná třída *HandlingWifiInfo*.

```
...

@Override
protected void onCreate() {
    existingFileName = new HandlingWifiInfo().getWifiSSID(
                            getApplicationContext()) + ".txt".replaceAll(" ", "_");

    checkIfFileExists(existingFileName);
}

...

private void checkIfFileExists(String fname) {
    File f = new File(getApplicationContext().getExternalFilesDir(null), fname);
    if(!f.exists())
        Log.d(TAG, "File " + fname + " NOT found -> New information about
                    current WiFi network is needed.");
    else
        new FileExistsDialog(this, fname).show();
}
```

Pokud již v úložišti takovýto soubor existuje, aplikace uživateli nabídne pomocí dialogu možnost přeskočit proces připojení k VPN společně s následnou výměnou dat s adresářovým serverem, neboť tyto data uživatel aplikaci poskytl již někdy dříve (předpokládá se, že data na adresářovém serveru vždy zůstanou). Druhou volbou, kterou dialog uživateli nabídne, je možnost vytvořit novou identitu i přesto, že pro danou síť byl uživatel vytvořen někdy dříve. Třetí možností je stav, kdy se dialog nezobrazí, což znamená, že požadovaný soubor v úložišti není. Tento stav je pro každou poprvé navštívenou bezdrátovou síť výchozí. Uživatel se tak musí připojit k VPN a získat z adresářového serveru novou identitu uživatele, kterou následně manuálně doplní o další požadované informace. Obrázek 4.3 nastiňuje grafické rozpoložení zmíněného dialogu společně s aktivitou *VPNActivity*.



Obrázek 4.3: *Grafická podoba aktivity, implementující podporu pro VPN (zleva: dialog, výchozí stav a stav po připojení se)*

4.5 Adresářové služby

Další oblastí, kterou se aplikace pro generování síťového provozu zabývá, je interakce s adresářovým serverem, do jehož lokální sítě se aplikace dostane pomocí VPN, jak je uvedeno v předešlé kapitole.

Adresářová služba mapuje názvy síťových prostředků na konkrétní síťové adresy a v intranetu a internetových aplikacích umožňuje sdílet informace a data mezi uživateli konkrétní sítě [42]. Data jsou zde reprezentována formou stromové hierarchie. Každý prostředek je v síti považován za objekt, kde informace se o něm ukládají jako tzv. atributy, které definují jeho vlastnosti. Pod jednotlivými objekty si lze představit například počítače, adresáře, soubory, uživatele, skupiny nebo tiskárny. Adresářové služby mohou neautorizovaným uživatelům k jednotlivým prostředkům omezit přístup. V podnikové sféře plní adresářové služby velmi důležitou roli, neboť poskytují sdílený přístup k prostředkům.

Soubor standardů, definující podobu adresářových služeb, vytvořily již v 80. letech organizace OSI a ITU. Jejich vzájemnou spoluprácí vznikl multiplatformní protokol LDAP (*Lightweight Directory Access Protocol*), spadající pod model TCP/IP. Poslední specifikací tohoto protokolu je LDAPv3, kterou popisuje doporučení RFC 4511.

LDAP standardně využívá pro nezabezpečenou komunikaci TCP port 389, kdežto TCP port 636 se standardně využívá při komunikaci po zabezpečeném kanále. Autentizace se pak řeší buďto jednoduchými ověřovacími mechanismy (anonymní či neověřená autentizace nebo autentizace pomocí jména/hesla). Další možnost je pak pomocí ověřovacího mechanismu SASL. Tato autentizační metoda navazuje na jiný mechanismus ověřování (např. *Kerberos*) [40].

S adresářovou službou se pojí pojem adresářový server, který takovouto službu poskytuje.

Vzájemná interakce mezi klientem a adresářovým serverem je v rámci LDAP založena na požadavcích klienta, který server žádá o operace pro dané adresáře – vyhledávání, vytváření, mazání, aktualizace dat atp. Ten na tyto požadavky reaguje a jejich výsledky zasílá klientovi zpět.

LDAP jakožto standardizovaný aplikační protokol slouží jako referenční bod pro všechny implementace adresářových serverů a nepředstavuje tedy jejich konečné řešení. To spočívá až na konkrétních vývojářích, kteří cílí svá řešení na konkrétní operační systémy, skupiny zákazníků nebo se zaměřují na otevřenost těchto systémů se specifickými vlastnostmi. Mezi nejčastější řešení patří například *Active Directory*, *Apache Directory Server*, *OpenLDAP*, *Red Hat Directory* nebo *Sun Java System Directory Server*.

[39][41][43]

4.5.1 Implementace přístupu k adresářovému serveru

Do aplikace, o které tato práce pojednává, je implementace podpory pro adresářové služby klíčová. Důvodem implementace konkrétního řešení adresářové služby je skutečnost, že aplikace vyžaduje komunikaci s adresářovým serverem, ke kterému lze přistupovat po úspěšném připojení se do virtuální privátní sítě.

Adresářový server se v dané síti vyskytuje v podobě *Windows Serveru 2012 R2*. Z hlediska LDAP řešení je proto vhodnou volbou využít adresářovou službu *Active Directory* (dále jen AD) od společnosti *Microsoft*, která je jednou z nejvyužívanějších adresářových služeb dnešní doby a speciálně pro operační systém *Windows* se logicky jeví jako jasná volba.

Po rekapitulaci, že vývoj aplikace probíhá pro platformu Android v programovacím jazyce Java, v rámci kterého je potřeba opětovně přistupovat ke sdíleným prostředkům na Windows Serveru pomocí AD, se nabízí jako řešení ve vývojovém prostředí *Android Studio* knihovna *JCIFS*. Tato knihovna je open source řešením komunikačního protokolu SMB, který je součástí relační vrstvy modelu OSI a poskytuje sdílený přístup k prostředkům v síti. Knihovna *JCIFS* v sobě zahrnuje podporu pro tento protokol, čímž aplikaci umožňuje implementovat přístup k AD. Aplikace tak díky tomu může pracovat se soubory na síťovém serveru s OS *Windows*.

Pro využití funkcí této knihovny je potřeba ji zahrnout do projektu. To je realizováno stáhnutím souboru *jcifs-1.3.19.jar* z [44]. Tento soubor se do aplikace importuje pomocí sestavovacího nástroje *Gradle*, kde do souboru *build.gradle* se do sekce *dependencies* připíše cesta k tomuto souboru. Následně je nutné provést synchronizaci tohoto souboru pro aplikování změn.

```
dependencies {
        ...
        implementation files('../libs/jcifs-1.3.19.jar')
        ...
}
```

Po zakomponování této knihovny do aplikace je možné využít jejích nástrojů a metod pro připojení se k adresářovému serveru na síti.

Na serveru se vyskytují 2 soubory v textové podobě, s nimiž aplikace pracuje.

4.5.1.1 ***Získání identity uživatele pro konkrétní Wi-Fi síť***

Obsahem prvního souboru je seznam uživatelů s heslem, kde každý řádek tohoto seznamu odpovídá právě jednomu uživateli. Jméno a příjmení každého uživatele z tohoto seznamu vychází z implementace nadřazeného projektu (zmíněného v kapitole 4.2), v němž byla tato dvojice vygenerována pomocí seznamů nejpoužívanějších českých jmen a příjmení. Podobně je to i s heslem, které je pro každého takto vytvořeného uživatele vygenerováno pomocí náhodných znaků v pevně daném formátu. Struktura souboru tedy vypadá následovně:

```
JIRI,NOVAK,A4Dbv%42
JIRI,SVOBODA,B0Xwd/35
```

Úkolem aplikace je vyjmout z tohoto seznamu uživatelů vždy první řádek, jehož obsah bude reprezentovat identitu uživatele. Aplikace později generuje síťový provoz na konkrétní Wi-Fi síti právě pod touto identitou.

Jelikož *Android Studio* neumožňuje vykonávat síťové operace v hlavním vlákně aplikace (*UI thread*), pro připojení k serveru se musí vytvořit vlákno vlastní, které pracuje na pozadí. K těmto účelům slouží třída *GetAdUser*, která dědí všechny veřejné metody jejího předka (třídy *AsyncTask*), čímž má možnost je nadeklarovat a překrýt podle sebe.

```
public class GetAdUser extends AsyncTask<String, Integer, String> {
    private AdListener adListener;

    ...

    public GetAdUser(AdListener adListener) {
        this.adListener = adListener;
    }

    @Override
    protected String doInBackground(String... params) {
        String user = "";
        String username = "...";
        String password = "...";
        String url = "smb://10.11.12.88/sourceusers/JmenaPrijmeniHeslaOK.txt";
        SmbFile file;
```

V metodě *doInBackgroud* jsou deklarovány následující proměnné: přihlašovací jméno s heslem pro přístup k adresářovému serveru, absolutní cesta k požadovanému souboru (včetně uvedení lokální IP adresy AD) a také proměnná *user*, do které se později ukládá výsledek.

```
        try {
            NtlmPasswordAuthentication auth =
                        new NtlmPasswordAuthentication(null, username, password);

            file = new SmbFile(url, auth);
```

Dále se v této metodě realizuje samotné připojení, kdy pomocí tříd *NtlmPasswordAuthentication* a *SmbFile* se aplikace vůči serveru autentizuje a následně se k němu připojí. Tyto metody poskytuje právě knihovna *JCIFS*.

```
            BufferedReader br = new BufferedReader(new InputStreamReader(
                                                new SmbFileInputStream(file)));

            StringBuilder inputBuffer = new StringBuilder();
```

```
            String line;

            while ((line = br.readLine()) != null) {
                inputBuffer.append(line);
                inputBuffer.append('\n');
            }
            br.close();

            String[] lines = inputBuffer.toString().split("\n");
            user = lines[0];
            BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(
                                            new SmbFileOutputStream(file)));

            for (int i = 1; i < lines.length; i++) {
                bw.write(lines[i], 0, lines[i].length());
                bw.newLine();
            }
            bw.close();

        } catch(Exception e) {
            e.printStackTrace();
        }
        return user;
    } // doInBackground
```

Po úspěšném připojení se provádí čtení požadovaného souboru po řádcích, které se ukládají do bufferu. Obsah prvního řádku slouží jako výstup metody, neboť se s ním bude dále pracovat. Po jeho získání je však nutné jej ze souboru odmazat. Toho je docíleno právě oním bufferem, který zapisuje do daného souboru data znovu, avšak s tím, že se začíná od druhé položky.

```
    @Override
    protected void onPostExecute(String result) {
        super.onPostExecute(result);
        adListener.setAdUser(result);
    }
} // class GetAdUser
```

V metodě *onPostExecute* je nakonec volána metoda *setAdUser*, kterou poskytuje rozhraní *AdListener*. Tato metoda předává data z vlákna do aktivity *LdapActivity*, která tuto metodu musí implementovat. Tato aktivita je v této kapitole popsána později.

Tímto je tedy popsáno, jak aplikace přistupuje k adresářovému serveru a ze souboru, který je na něm uložen, získává data, vyskytující se na prvním řádku. Ta obsahují identitu uživatele, pod kterou aplikace později generuje síťový provoz.

4.5.1.2 ***Uložení identity uživatele s doplňujícími informacemi o aktuální Wi-Fi síti***

Druhý soubor, se kterým aplikace pracuje, se rovněž vyskytuje na adresářovém serveru v síti. Slouží k ukládání identity uživatele, kterou aplikace získala z předchozí kapitoly. Identitu je však nutné doplnit o informace, vztahující se k bezdrátové síti, ke které je momentálně aplikace připojena.

Bez tohoto kroku by se samotné generování síťového provozu pro konkrétní Wi-Fi síť pod danou identitou obešlo, nicméně v kompetenci nadřazeného projektu by nebylo možné shromažďovat informace o konkrétních Wi-Fi sítích, ve kterých generování síťového provozu probíhalo. Pokud by útočník získal přístup k nezabezpečeným přihlašovacím údajům a připojil se s nimi na konkrétní servery, nebylo by možné zjistit, ze které Wi-Fi sítě se přihlásil. Pokud by tak aplikace využívala stejné přihlašovací údaje (identitu) ve více bezdrátových sítích, nebylo by možné dohledat, ze které konkrétní

sítě se útočník přihlásil. Potom, co aplikace shromáždí všechny požadované informace a uloží je do konkrétního souboru na *Windows* serveru, se o ně dále postará již dříve zmíněný *PowerShell* skript, který na jejich základě vytvoří na AD nového uživatele.

Informace mají následující formát:

Tabulka 4.1: *Informace, definující uživatele na Active Directory*

FirstName	LastName	Password	Title	Description
Office	SSID	MAC	DeviceIP	NetworkIP
NetworkInfo	GpsLocation	WifiPassword	Date	Division

Kde:

- FirstName, LastName, Password: Identita uživatele (viz předchozí kapitola)
- Title, Description: Stejné hodnoty jako pole LastName
- Office: Popis umístění sítě (nutný vložit manuálně)
- SSID: Název sítě
- MAC: MAC adresa přístupového bodu sítě
- DeviceIP: Lokální IP adresa, kterou přidělí síť zařízení
- NetworkIP: Veřejná IP adresa sítě
- NetworkInfo: Vlastní název pro Wi-Fi síť (nutný vložit manuálně)
- GpsLocation: Aktuální GPS souřadnice, tzn. GPS umístění sítě
- WifiPassword: Přístupové heslo k síti (nutno vložit manuálně)
- Date: Datum vytvoření
- Division: Označení vývojáře

Při vyplnění všech informací by mohl výstup vypadat například takto:

Tabulka 4.2: *Příklad poskytnutých informací, definující uživatele na Active Directory*

JIRI	Novak	A4Dbv%42	NOVAK	NOVAK
Supermarket	OVAPublic	00:10:65:C0:00:01	10.11.0.105	1.2.3.4
Obchod XY	49.83N18.16E	heslosite	2021-04-30	FRE0025

Obdobně jako v minulé kapitole se i zde využívá síťových operací, tudíž je opět nutné využít vlákna, běžícího na pozadí. Třída, která vystupuje jako potomek třídy *AsyncTask*, se v tomto případě jmenuje *UploadAdUser*.

```
public class UploadAdUser extends AsyncTask<Void, Void, Void> {
    private Context c;
    private AdListener adListener;
    private AdUser adUser;

    ...

    public UploadAdUser(Context c, AdListener adListener, AdUser adUser) {
        this.c = c;
        this.adListener = adListener;
        this.adUser = adUser;
    }

    @Override
    protected Void doInBackground(Void... voids) {
        ...
```

```
        String url = "smb://10.11.12.88/share/test/export.csv";
        try {
            ...
            BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(
                                                    new SmbFileOutputStream(file)));

            bw.write(adUser.getAdFileContent());
            bw.close();
        } catch(Exception e) {
            e.printStackTrace();
        }
        createLocalFile();
        return null;
    }
```

Konstruktor této třídy obsahuje 3 parametry. Jedná se kontext aplikace, instanci rozhraní *AdListener* a instanci třídy *AdUser*. Přístup k AD se zde řeší stejným způsobem jako ve třídě *GetAdUser*, která byla popsána v minulé kapitole. Po připojení se k serveru třída zapisuje do souboru informace o identitě uživatele s údaji o síti pomocí třídy *BufferedWriter*. Tato data získá z veřejné metody *getAdFileContent*, která je deklarována třídou *AdUser*.

```
    @Override
    protected void onPostExecute(Void v) {
        super.onPostExecute(v);
        adListener.adUserUploadDone();
    }

    private void createLocalFile() {
        String fileNameSsid = adUser.getSSID();
        fileNameSsid = fileNameSsid.replaceAll(" ", "_");
        String con = adUser.getAdFileContent();

        try {
            new HandlingWifiInfo().writeToFile(c, fileNameSsid + ".txt", con);
        } catch (IOException e) {
            e.printStackTrace();
            Log.d(TAG, "Error writing to file " + fileNameSsid + ".txt");
        }
    }
} // class UploadAdUser
```

Dalším krokem je vytvoření souboru se stejným obsahem i v lokálním úložišti zařízení - metoda *createLocalFile*. S tímto souborem aplikace pracuje jednak v další aktivitě a jednak v aktuální aktivitě *LdapActivity*, ze které byla celá tato třída volána. Nakonec třída využije metodu *adUserUploadDone*, která je součástí rozhraní *AdListener*, čímž informuje aktivitu *LdapActivity* o dokončení celého procesu.

4.5.1.3 ***Aktivita, implementující adresářové služby***

Závěrem této podkapitoly je vhodné objasnit funkčnost aktivity, která v sobě zahrnuje implementaci předchozích dvou podkapitol 4.5.1.1 a 4.5.1.2. Aplikace je pojmenována *LdapActivity* a je volána po úspěšném připojení aplikace k VPN.

```
public class LdapActivity extends AppCompatActivity implements
View.OnClickListener, AdListener {
    private String fileName;
    private String user;
    private GetAdUser getAdUser;
    private TextView nameOfUserInfo, nameOfUser, pw;

    ...
```

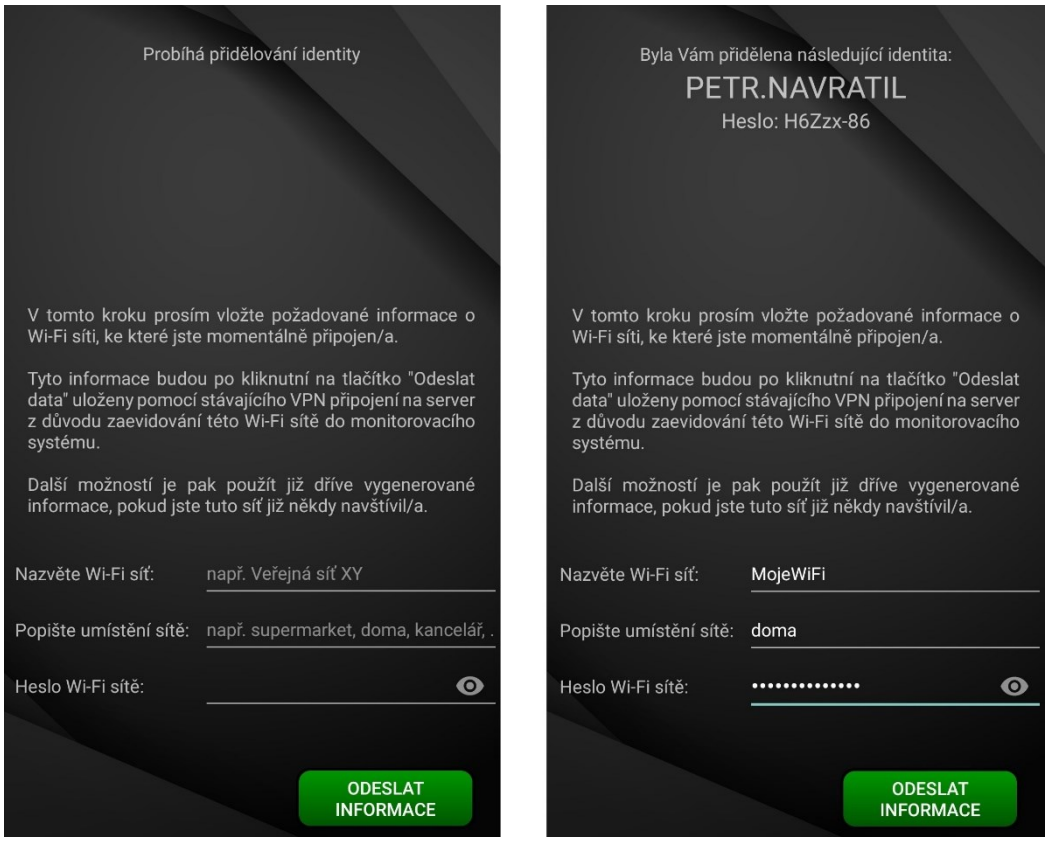
```
    @Override
    protected void onStart() {
        super.onStart();
        registerReceiver(finishReceiver, new IntentFilter("FinishActivity"));
        getAdUser = new GetAdUser(this);
        getAdUser.execute();
    }
    @Override
    public void setAdUser(String user) {
        this.user = user;
        String[] u = user.split(",");
        nameOfUserInfo.setText(R.string.infoGeneratedIdentity);
        nameOfUser.setText(u[0] + "." + u[1]);
        pw.setText("Heslo: " + u[2]);
    }
```

Prvním krokem je registrace lokálního *broadcast receiveru* při samotném spuštění aktivity. Ten se obdobně jako u předchozích aktivit zavolá v případě, kdy zařízení ztratí přístup k Wi-Fi síti, čímž se *LdapActivity* ukončí a uživatel se vrátí do úvodní aktivity celé aplikace. Rovněž se spustí proces získání identity uživatele pomocí třídy *GetAdUser*, který běží na pozadí.

Aktivita musí dále definovat obsah metod, které jsou součástí rozhraní *AdListener*, neboť toto rozhraní sama implementuje. Jedná se o metody *setAdUser* a *adUserUploadDone.* Metoda *setAdUser* je volána po ukončení tohoto procesu a jejím smyslem je vypsat získaný výsledek na obrazovku do odpovídajících textových polí.

```
    @Override
    public void onClick(View view) {
        if (view.getId() == R.id.btnSendData) {
            btnSendData.setEnabled(false);
            btnSendData.setText(R.string.sendingInformation);

            adUser = new AdUser(user,
                                HandlingWifiInfo.readFromFile(
                                     getApplicationContext(),"wifi_info.txt"),
                                editTextInfo.getText().toString(),
                                editTextOffice.getText().toString(),
                                editTextWifiPassword.getText().toString());

            uploadAdUser = new UploadAdUser(this, this, adUser);
            uploadAdUser.execute();
        }
    }

    ...

    @Override
    public void adUserUploadDone() {
        goToNextActivity();
    }
} // class LdapActivity
```

Po poskytnutí požadovaných informací uživatelem a kliknutí na tlačítko „Odeslat informace“ se spustí proces uložení výsledné identity uživatele v požadovaném formátu do souboru na AD. Zpracování dat do požadovaného formátu provádí pomocná třída *AdUser*, jejíž instance se dále použije jako parametr konstruktoru třídy *UploadAdUser*. Metoda *adUserUploadDone* pak volá metodu *goToNextActivity*, která řeší přechod do další aktivity prostřednictvím explicitního intent objektu. Z pohledu uživatele pak vypadá výsledná podoba aktivity *LdapActivity* následovně:



Obrázek 4.4: *Aktivita, implementující podporu pro adresářové služby (vlevo úvodní stav, vpravo po vygenerování identity uživatele a uvedení příslušných informací)*

4.5.2 **Konfigurace na straně Active Directory**

Jak již bylo zmíněno v kapitole 4.2, na LDAP serveru je spuštěn *PowerShell* skript, jehož úkolem je hlídat změny na souborovém systému pro konkrétní adresář. Pokaždé, když tedy aplikace do tohoto adresáře vloží soubor s informacemi, definující identitu uživatele, skript na tento stav zareaguje tak, že ze souboru získá jednotlivé položky a na jejich základě vytvoří na AD uživatele.

Kromě vytvoření uživatele na AD byl do skriptu připsán způsob, jak pro tohoto uživatele vytvořit odpovídající koncový bod na SIP serveru, pod kterým se bude uživatel později registrovat na *Asterisk* v rámci generování síťového provozu. Detailnějšímu popisu tohoto způsobu se věnuje kapitola 4.6.5.4, pojednávající o generování síťového provozu pro protokol SIP.

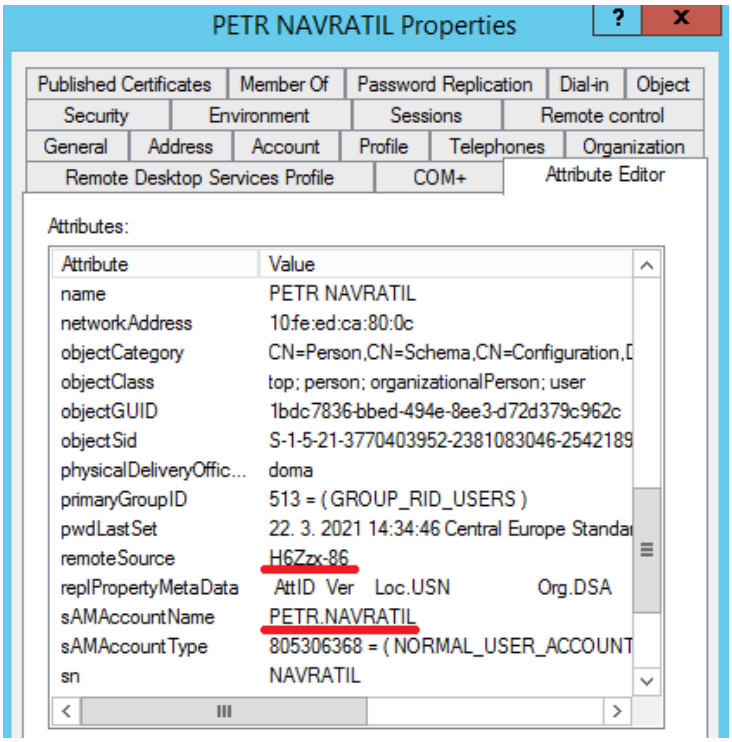
Po vytvoření uživatele na AD se nakonec soubor, z něhož čtení probíhalo, vymaže, aby nedocházelo k případným nechtěným konfliktům souboru při opětovném nahrávání s daty nového uživatele.

Na následujícím obrázku 4.5 je možno vidět obsah zmíněného skriptu.

```powershell
Import-Module ActiveDirectory

$watcher = New-Object System.IO.FileSystemWatcher
$watcher.Path = "C:\share\test"
$watcher.Filter = "*.*"
$watcher.IncludeSubdirectories = $true
$watcher.EnableRaisingEvents = $true

$action = {
    $ADUsers = Import-Csv C:\share\test\export.csv -Delimiter ","

    foreach ($User in $ADUsers) {
        $password = $User.Password

        if (Get-ADUser -F { SamAccountName -eq "$($User.First).$($User.Last)" }) {
            Write-Warning "Account already exists."
        }
        else {
            $Attributes = @{
                SamAccountName = "$($User.First).$($User.Last)"
                UserPrincipalName = "$($User.First).$($User.Last)@[redacted]"
                Name = "$($User.First) $($User.Last)"
                GivenName = "$($User.First)"
                Surname = "$($User.Last)"
                Enabled = $True
                DisplayName = "$($User.Title)"
                Path = "OU=services,DC=[redacted]"
                Description = "$($User.Description)"
                Office = "$($User.Office)"
                OtherAttributes = @{mobile = "$($User.Mobile)";
                                    networkAddress = "$($User.networkAddress)";
                                    ipHostNumber = "$($User.ipHostNumber)";
                                    ipNetworkNumber = "$($User.ipNetworkNumber)";
                                    info = "$($User.info)";
                                    destinationIndicator = "$($User.destinationIndicator)";
                                    host = "$($User.host)";
                                    remoteSource = "$($User.Password)";
                                    division = "$($User.division)"}
                AccountPassword = (ConvertTo-secureString $password -AsPlainText -Force)
                ChangePasswordAtLogon = $False
            }

            New-ADUser @Attributes
            Write-Host "Account" $Attributes.SamAccountName "created." -ForegroundColor Cyan

            #Asterisk
            $string = "[$($Attributes.SamAccountName)]$'\n'
                       type=endpoint$'\n'
                       context=from-internal$'\n'
                       allow=!all,alaw,g722$'\n'
                       aors=$($Attributes.SamAccountName)$'\n'
                       auth=$($Attributes.SamAccountName)$'\n'$'\n'
                       [$($Attributes.SamAccountName)]$'\n'
                       type=aor$'\n'
                       max_contacts=1000$'\n'$'\n'
                       [$($Attributes.SamAccountName)]$'\n'
                       type=auth$'\n'
                       auth_type=userpass$'\n'
                       username=$($Attributes.SamAccountName)$'\n'
                       password=$($password)"
            $endpoint = $string -replace '\s',''
            $sam = $($Attributes.SamAccountName)
            $sam = $sam -replace '\s',''

            plink -ssh [redacted] -pw [redacted] "echo $($endpoint) > /etc/asterisk/ad_endpoints/$($sam).conf"
            plink -ssh [redacted] -pw [redacted] service asterisk reload
        }
    }
    Remove-Item C:\share\test\export.csv
}
Register-ObjectEvent $watcher "Created" -Action $action
while ($true) {sleep 5}
```

Obrázek 4.5: *PowerShell skript, vytvářející Active Directory uživatele*

Pokud vytvoření uživatele proběhlo úspěšně, pomocí nástroje *Active Directory Users and Computers* si lze zobrazit veškeré jeho vlastnosti a nastavené parametry (viz obrázek 4.6).



Obrázek 4.6: *Vytvořený uživatel na Active Directory (Windows Server 2012 R2)*

Další využití najdou v rámci vyvíjené aplikace zejména položky *sAMAccountName* (uživatelské jméno) a *remoteSource* (heslo), pomocí kterých se bude aplikace přihlašovat k dříve zmíněným serverům a právě tyto dvě položky budou dohledatelné případným útočníkem při eventuálním monitoringu síťového provozu.

4.6 Generování síťového provozu

Po úspěšném vytvoření uživatele na AD a získání odpovídajících přihlašovacích údajů se lze v aplikaci přesunout k části, věnující se již samotnému generování síťového provozu. Jak již bylo nastíněno dříve, smyslem generování síťového provozu je v této práci simulovat chování běžného uživatele na internetu v určitém předem daném časovém intervalu. V praxi to pak znamená následující chování:

Uživatel si nejprve zapne webový prohlížeč a navštíví domovskou webovou stránku (dejme tomu zpravodajský portál). Síťový provoz, který byl během tohoto procesu vytvořen, se skládá z požadavku uživatele na webový server o zaslání obsahu požadované stránky a z odpovědi tohoto serveru zpět uživateli s požadovaným obsahem (HTTP/S požadavek a odpověď). Aplikace vystupuje v roli uživatele, což znamená, že pokud by měla tento síťový provoz generovat, stačí jí poslat požadavek na konkrétní webový server a o zpětnou vazbu (odpověď serveru zpět uživateli) se postará samotný server. Na domovské stránce zůstane uživatel do doby, než najde zpravodajský článek, který ho zajímá a následně klikne na jeho odkaz, čímž vygeneruje opět síťový provoz. Během několika minut tak uživatel zpravidla navštíví několik webových stránek, mezi jejichž zobrazeními jsou určité časové rozestupy v závislosti na charakteristice konkrétní stránky.

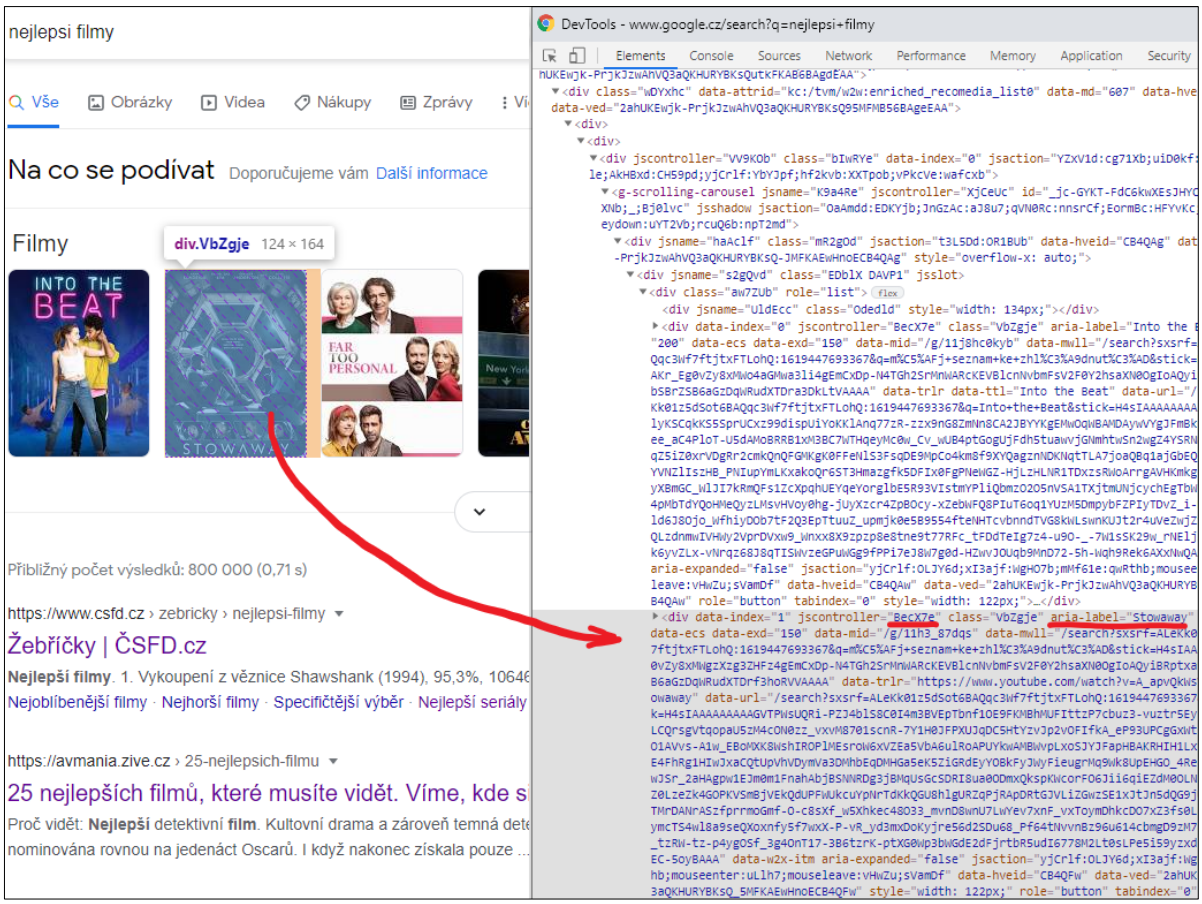
Nyní však vyvstává otázka, jak lze toto chování naimplementovat v aplikaci. První možností je na základě modelů síťového provozu (viz kapitola 1.2). Další možností je pak analyzovat provoz reálné

sítě a na základě dříve získaných vzorků síťového provozu zkonstruovat provoz nový, o čemž pojednává kapitola 1.3.

4.6.1 Analýza obsahu webových stránek

Z těchto teoretických poznatků a charakteristiky práce (aplikace má simulovat chování běžného uživatele) vyplývá, že jednou z možností implementace pro platformu *Android* může být sestavení vlastního seznamu odkazů na webové stránky (URL adres), se kterými aplikace následně komunikuje. Pro zajištění logických souvislostí mezi webovými stránkami je nutné nejprve manuálně analyzovat obsah HTML dokumentů a najít jednotlivé elementy, nesoucí URL adresy na související webové stránky nebo získat textový řetězec, který lze následně využít v jiné URL adrese.

V rámci této práce byla takováto analýza provedena v nástroji *DevTools*, který je součástí webového prohlížeče *Google Chrome*. Na obrázku 4.7 lze vidět příklad analýzy webové stránky www.google.cz/search?q=nejlepsi+filmy, která je na seznamu stránek, se kterými aplikace komunikuje. Postupným nalezením cesty k danému elementu se dají vyhledat požadované informace, se kterými se dá dále pracovat. V tomto případě se jedná o nalezení názvu filmu, který bude později použit jako parametr další URL adresy.



Obrázek 4.7: *Manuální vyhledání konkrétní informace pomocí nástroje DevTools*

4.6.2 Práce s obsahem webových stránek

Toto prohledávání jednotlivých částí HTML dokumentu lze v prostředí *Android Studio* realizovat pomocí knihovny *JSOUP.* Její naimportování do projektu lze vyřešit skrze sestavovací nástroj *Gradle* přidáním následujícího řádku do sekce *dependencies*:

```
dependencies {
        ...
        implementation 'org.jsoup:jsoup:1.13.1'
        ...
}
```

Po zakomponování této knihovny do aplikace je možné využít jejích nástrojů a metod pro získání dat z HTML dokumentů.

Ukázka zdrojového kódu níže demonstruje získání názvu filmu (dle obrázku 4.7) na základě předchozí analýzy stromového uspořádání jednotlivých elementů v daném HTML dokumentu. Tento název je dále použit jako parametr další URL adresy.

```
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
...

Document doc = new Document("empty");
Random rn = new Random();

doc = website(doc, "https://www.google.cz/search?q=nejlepsi+filmy", 1.8);
String movieName = doc.getElementsByAttributeValue("jscontroller","BecX7e")
                      .get(8)
                      .attr("aria-label");

movieName = StringUtils.stripAccents(movieName);
movieName = movieName.replaceAll(" ","+").replaceAll("[()]","");
doc = website(doc, "https://www.csfd.cz/hledat/?q=" + movieName, 1.0);
```

Po deklarování potřebných instancí tříd *Document* a *Random* je volána metoda *website*, jejímž úkolem je samotná návštěva webové stránky na dané URL adrese. Tato metoda vrací zpět HTML dokument, který je čitelný v textové podobě. Nyní je nutné na tento dokument zavolat několik metod, díky kterým se dá v jeho stromové struktuře dostat až k samotnému názvu filmu. Knihovna *JSOUP* nabízí ve svém repertoáru mnoho metod, díky kterým se dá v HTML dokumentech orientovat. Mezi využité metody této knihovny patří v tomto případě nejprve *getElementsByAttributeValue*, která přijímá 2 parametry – atribut HTML elementu (tagu) a název tohoto atributu. V případě atributu *jscontroller* a jeho názvu „*BecX7e*“ by se do proměnné *movieName* jako textový řetězec uložil veškerý obsah elementu *div* s tímto názvem. Parametrem metody *get* je pak index elementu s daným názvem „*BecX7e*“. Díky tomu, že daný dokument obsahuje tento název 8x (zobrazuje 8 různých filmů), vybírá se pouze jeden konkrétní. Toho je docíleno vygenerováním náhodného čísla v rozmezí 0-7. Metodou *attr* se specifikováním konkrétního atributu se lze nakonec dostat přímo k samotnému názvu filmu, jehož textový řetězec tato metoda vrátí.

Následuje jednoduchá úprava tohoto řetězce tak, aby neobsahoval diakritiku a rovněž dojde k odstranění problematických znaků jako mezera či závorky, které se do URL adres nehodí.

Takto získaný název filmu lze dále využít jako součást další URL adresy, která je opět parametrem metody *website*, jejíž obsah přichází na řadu nyní.

```
private Document website(Document doc, String url, double timeConst) {
    if(isCancelled())
        return null;

    try {
        if(doc == null)
            Jsoup.connect(url);
        else
            doc = Jsoup.connect(url).get();
        this.url = url;
    } catch (IOException e) { e.printStackTrace(); }

    publishProgress(progress++);

    if(!fastGenerating) {
        double st = (double)avgWaitTime * timeConst;
        waitTime = (long)st;

        try {
            Thread.sleep(waitTime);
        } catch (InterruptedException e) { e.printStackTrace(); }
    }
    return doc;
}
```

Jádrem samotné metody je příkaz *„doc = Jsoup.connect(url).get();“*, který odesílá požadavek o zobrazení HTML dokumentu na dané URL adrese. Následuje čekání na načtení obsahu tohoto dokumentu, který se následně uloží do lokální proměnné.

4.6.3 Simulace chování běžného uživatele

Další důležitý prvek metody *website* se týká času – konkrétně pozastavení běžícího procesu na předem danou dobu – *waitTime*. Smysl tohoto pozastavení vychází opět ze snahy simulovat chování běžného uživatele. Uživatel se totiž na webových stránkách nevyskytuje v řádě milisekund, nicméně trvá mu nějakou dobu, než klikne na odkaz, který ho přesune na stránku jinou. Právě z tohoto důvodu je po každé návštěvě webové stránky nastavena pauza. Její délka vychází z konstanty (proměnná *timeConst*), která byla stanovena na základě logického odhadu při porovnání s ostatními webovými stránkami ze seznamu. Konstanta nabývá hodnot v rozmezí 0 - 2, přičemž průměrné době čekání odpovídá hodnota 1. Průměrná doba čekání (proměnná *avgSleepTime*) je stejná pro všechny URL adresy a vypočítává se následovně:

$$avgWaitTime = \left(T - \sum_{n=0}^{n} AVG_{T_n}\right)/N$$

Kde:

- T je celkový minimální čas (nastavený uživatelem), po který bude generování síťového provozu probíhat,
- AVG_{T_n} je průměrný čas, během kterého se načte obsah webové stránky n a během kterého se uloží odpovídající HTML dokument do lokální proměnné,
- N je celkový počet webových stránek v seznamu.

V metodě *website* si lze všimnout i klíčového slova *Thread*, které napovídá, že celý proces probíhá ve vláknech, aby nebyla omezena funkčnost hlavního vlákna, na kterém se mimo jiné vykresluje grafika. Součástí metody je rovněž dotaz *„if(isCancelled())*“, který rovněž odkazuje na

operaci s vláknem a souvisí s manuálním zastavením celého procesu generování síťového provozu uživatelem, o němž se zmiňuje kapitola 4.6.6.

Webové stránky, které spolu logicky souvisejí na základě uživatelova chování jsou přiřazeny do jedné konkrétní skupiny. Aplikace tedy díky tomuto principu obsahuje několik takovýchto skupin, kde každá obsahuje URL adresy, odkazující se na dané stránky, které spolu úzce souvisejí nebo na sebe logicky navazují. Těchto stránek je předem daný počet a aplikace je musí během celkového minimálního času, stanoveného uživatelem, navštívit (ve smyslu webového prohlížeče). Čas je záměrně uváděn jako minimální, neboť je ve výsledku navýšen o dobu, během které se načte obsah jednotlivých stránek, což je proměnná veličina, která se odvíjí jednak od rychlosti internetového připojení a jednak od samotné rychlosti zpracování požadavků daných serverů. To samé platí i při interakci s dříve zmíněnými konfigurovanými servery, kde například u FTP serveru se čeká na nahrání souboru. Mezi navštěvováním konkrétních webových stránek aplikace generuje síťový provoz kromě zmíněného HTTP i pro protokoly FTP, SIP, IMAP, POP3 a SMTP.

4.6.4 Nastavení parametrů generování síťového provozu

Smyslem aktivity, která se stará o nastavení parametrů generování síťového provozu, je nabídnout uživateli grafické rozhraní, díky kterému může uživatel přehledně nastavit vstupní parametry pro následné generování síťového provozu. Jmenovitě se jedná o nastavení minimálního času, po který bude generování probíhat a také výběr jednotlivých protokolů, kde v jejich režii bude aplikace provádět interakci s odpovídajícími servery. Tento čas je pro praktickou demonstraci stanoven do rozmezí 1 až 10 minut s tím, že lze zadat i v sekundách. Pro praktičnost byly do této aktivity přidány opět informace o přihlašovacích údajích, které budou při následném generování provozu pro danou bezdrátovou síť využity.

Co se týče zdrojového kódu, lze v této aktivitě zmínit například chování jednotlivých tlačítek, které odpovídají jednotlivým protokolům a výběru času v sekundách či minutách. V rámci *Android Studia* se totiž nejedná o klasická tlačítka (komponenta *Button*), nicméně o *ToggleButton*, která nabývají dvou stavů. Ke zjištění těchto stavů lze využít metodu *isChecked* nebo definovat naslouchač události (listener).

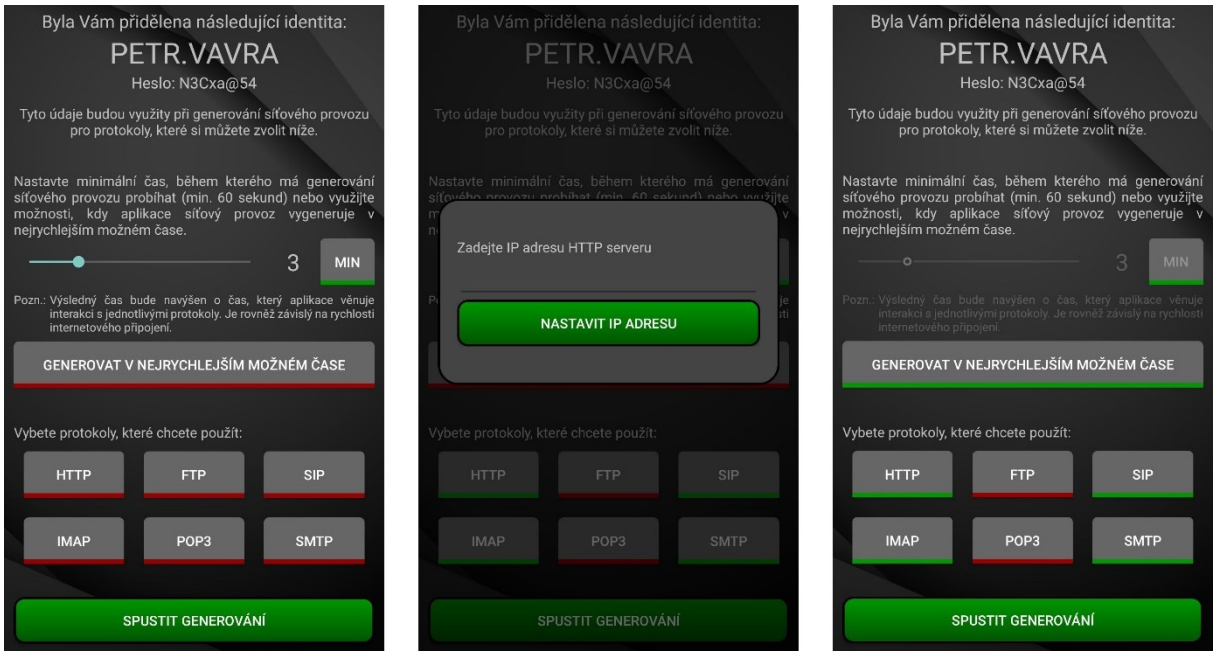
```
boolean[] selectedProtocols;
ToggleButton toggleHttp;

selectedProtocols[0] = toggleHttp.isChecked();

toggleHttp.setOnCheckedChangeListener((compoundButton, checked) -> {
    if(checked) {
        toggleHttp.setBackgroundResource(R.drawable.selected_button);
        selectedProtocols[0] = true;
    } else {
        toggleHttp.setBackgroundResource(R.drawable.unselected_button);
        selectedProtocols[0] = false;
    }
});
```

Dalším bodem, který je pro tato tlačítka (a samozřejmě i pro celou aplikaci) důležitý, je jejich grafická podoba. Ta je možná nastavit pro jednotlivé stavy tlačítka například nastavením pozadí, kterému odpovídá vlastní XML soubor, který lze vytvořit v *Android Studiu*. Pro tlačítko *toggleHttp* ve stavu *checked* má tento soubor následující podobu:

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
   <item>
      <shape android:shape="rectangle" >
         <corners android:topLeftRadius="5dp" android:topRightRadius="5dp" />
         <solid android:color="#090" />
      </shape>
   </item>
   <item android:bottom="5dp">
      <shape android:shape="rectangle" >
         <solid android:color="#666" />
         <corners android:topLeftRadius="5dp" android:topRightRadius="5dp" />
      </shape>
   </item>
</layer-list>
```

Mezi dodatečné funkce, které byly do této aktivity implementovány, patří nejprve možnost nastavit u jednotlivých protokolů IP adresu serveru, na kterém je nakonfigurována odpovídající služba, která má nastaveno ověřování uživatelů vůči AD. Po uvedení odpovídajícího protokolu, respektive tlačítka do stavu „vybrán“, se uživateli zobrazí dialog, do kterého může IP adresu zadat. Pro každý protokol je nastavena výchozí IP adresa, nicméně uživatel si ji může podle své potřeby změnit. Grafická podoba této aktivity je pak zřejmá z následujícího obrázku 4.8.

Druhou funkcí, která se jeví jako velmi užitečná zejména při samotném testování aplikace, je možnost ignorovat nastavení času, který uživatel stanoví. Místo tohoto času se využije alternativní možnosti, kdy aplikace vygeneruje síťový provoz v nejrychlejším možném čase. To znamená, že zruší čekací dobu po návštěvě každé webové stránky.



Obrázek 4.8: *Grafická podoba aktivity pro nastavení parametrů generování*

Po kliknutí na tlačítko „Spustit generování“ se spustí samotný proces generování síťového provozu, který vyvolá dialog, jemuž jsou předána potřebná data – konkrétně se jedná o stanovený čas, pole hodnot pravda/nepravda, definující stav vybraných protokolů, pole IP adres serverů a také přihlašovací údaje, které jsou v této aktivitě již zobrazeny.

4.6.5 Generování síťového provozu pro vybrané protokoly

Na řadu nyní přichází část, pojednávající o jednotlivých protokolech aplikační vrstvy referenčního modelu OSI dle zadání práce, pro které je v aplikaci potřeba generovat síťový provoz. Pro každý z těchto protokolů (HTTP, FTP, SIP, IMAP, POP3 a SMTP) byl nejprve nakonfigurován odpovídající server tak, aby umožňoval ověřování vůči LDAP/AD. Každému z těchto serverů je aplikace schopna poskytnout přihlašovací údaje, které odpovídají platnému uživateli na Active Directory. To aplikace zajišťuje implementací odpovídajících klientů, kteří jsou schopni se k těmto serverům připojit.

4.6.5.1 ***HTTP***

K posílání uživatelských dat (v tomto případě přihlašovacích údajů) slouží v protokolu HTTP metoda POST, která se rovněž využívá při poskytnutí přihlašovacích údajů HTTP serveru v této práci. Touto metodou se obvykle posílají data z webových formulářů, která by se měla vždy posílat zašifrovaná, tudíž by se měl standardně využívat protokol HTTPS. V této práci je však žádoucí posílat přihlašovací údaje na daný server bez jakéhokoliv zabezpečení, tudíž se využije prosté HTTP.

Na reálném serveru, který byl poskytnut vedoucím této práce, je nainstalován HTTP server *Apache*, který je nejrozšířenějším open source webovým serverem vůbec. Základní instalační balíček *apache2* v rámci unixových systémů neobsahuje podporu SSL, což je pro potřeby této práce ideální. Jednoduchou možností, jak si lze ověřit, zda konkrétní *virtuální host Apache* serveru nepodporuje HTTPS, je kontrola obsahu souboru daného hosta v adresáři */etc/apache2/sites-available/*.

Po zprovoznění *Apache* HTTP serveru je nutné vytvořit jednoduchou webovou stránku (HTML dokument), jejímž obsahem je formulář s tlačítkem pro odeslání požadavku, do kterého lze vepsat konkrétní přihlašovací údaje. Dále je nutné specifikovat odpovídající akci, která se stane po kliknutí na tlačítko po uvedení těchto údajů. V tomto případě je uživatel přesměrován na stránku, psanou ve skriptovacím jazyce *PHP*, která ověřuje zadané údaje vůči AD (zkrácená verze):

```
<?php

    $USER = $_POST[„username“];
    $PASSWORD = $_POST[„password“];
    $CONNECTION = ldap_connect(„ldap://IPadresaAD+port“);
    if($CONNECTION && !empty($USER) && !empty($PASSWORD)) {
        $USER .= „@ADdomena“;
        $LDAPBIND = ldap_bind($CONNECTION, $USER, $PASSWORD);
        if($LDAPBIND)
            success ...
    } else
    echo „Empty credentials“;

?>
```

Pokud jsou vložené přihlašovací údaje správné (existují na AD), uživatel byl úspěšně autorizován a tuto aktivitu lze na AD zaznamenat. Klíčové jsou v ukázce výše z pohledu LDAP ověřování právě funkce *ldap_connect* a *ldap_bind.*

Implementaci klientské strany protokolu HTTP lze pro *Java* aplikace řešit importem knihovny *OkHttp*. Jedná se o efektivní open source nástroj, vystupující v roli HTTP klienta. Umožňuje posílat jak synchronní, tak asynchronní HTTP požadavky. Do aplikace se naimportuje pomocí nástroje *Gradle* následovně:

```
dependencies {
        ...
        implementation 'com.squareup.okhttp3:okhttp:4.9.0'
        ...
}
```

K sestavení potřebného HTTP požadavku POST s přihlašovacími údaji je nutné definovat obsah tohoto požadavku. Ten se skládá z URL adresy, těla formuláře a hlaviček. Jelikož nakonfigurovaný HTTP server poskytnutí žádných povinných hlaviček nevyžaduje, je možné tyto hlavičky vynechat. Tělo formuláře pak obsahuje samotné přihlašovací údaje společně s identitou odpovídajících HTML elementů, do kterých tyto údaje v rámci HTML formuláře náleží. Tento požadavek je pak parametrem instance třídy *OkHttpClient*, na kterou se volá metoda *newCall* a *enqueue*, které zajistí poslání tohoto požadavku danému serveru. Nakonec je potřeba dostat ze serveru odpověď, která je v rámci aplikace naimplementována zpětným voláním (tzv. *callbackem*). Když odpověď ze serveru dorazí, tento *callback* se zavolá a pomocí *listeneru* vrátí tuto odpověď do nadřazené třídy tak, aby se uživateli zobrazila na obrazovce v reálném čase. Celý proces, zmíněný v tomto odstavci, je zřejmý z následujícího výňatku ze zdrojového kódu:

```
private final OkHttpClient client;
private final String host, username, password;
private final HttpResponse httpResponse;

...

RequestBody body = new FormBody.Builder()
                        .add("username", username)
                        .add("password", password)
                        .add("sub","Login").build();

Request request = new Request.Builder().url(host).post(body).build();

client.newCall(request).enqueue(new Callback() {
   @Override
   public void onFailure(@NonNull Call call, @NonNull IOException e) {
       e.printStackTrace();
       httpResponse.httpFinished("HTTP response failure");
   }
   @Override
   public void onResponse(@NonNull Call c,@NonNull Response r)throws IOException {
       httpResponse.httpFinished(r.body().string());
   }
});
```

4.6.5.2 ***FTP***

Dalším protokolem v řadě je *File Transfer Protocol*. Jak je patrné z jeho názvu, tento protokol umožňuje přenášet soubory v počítačových sítích. Pracuje na portech 20 a 21, kde první jmenovaný zajišťuje přenos dat a druhý slouží k řídícím procesům. FTP využívá z logických důvodů pouze spojově orientovaného protokolu TCP.

Z hlediska konfigurace byl na poskytnutém serveru vedoucím práce využit FTP server zvaný *Proftpd*. Aby se mohli na tento server přihlašovat uživatelé AD (respektive aby bylo možné využít přihlašovacích údajů, poskytnutých aplikací pro danou bezdrátovou síť), je nutné upravit obsah souboru *ldap.conf*, který se nachází v adresáři */etc/proftpd/.* Jeho obsah s jednotlivými poli, která je nutné nastavit, je zveřejněn na následující ukázce.

```
<IfModule mod_ldap.c>

   LDAPServer                    ldap://IP adresa:port/??sub
   LDAPBindDN                    "cn=Ondrej Freisler,cn=Users,dc=AD_doména"
                                 "Heslo"
   LDAPUsers                     "ou=services,dc=AD_doména"
                                 (sAMAccountName=%u)
   LDAPAttr uid                  sAMAccountName
   LDAPAttr gidNumber            primaryGroupID
   LDAPAttr homeDirectory        /ftpshare
   LDAPAuthBinds                 on
   LDAPGenerateHomedir           on 0755
   CreateHome                    on 0755
   LDAPGenerateHomedirPrefix     /ftpshare
   LDAPForceGeneratedHomedir     on
   LDAPLog                       /var/log/mod_ldap.log

</IfModule>
```

Po konfiguraci tohoto souboru je rovněž potřeba modul *mod_ldap.c* aktivovat. To je možné udělat v souboru *modules.conf* (opět v */etc/proftpd/*), ve kterém stačí odstranit komentář na začátku příslušného řádku, který odpovídá názvu tohoto modulu.

Pro zabezpečení slouží u *Proftpd* soubor *tls.conf*. Jelikož je jeho obsah ve výchozím stavu zakomentovaný, není zde potřeba nic upravovat.

Implementaci FTP klienta v aplikaci řeší knihovna *Apache Commons*, jejíž binární soubor lze stáhnout z oficiálních stránek [45]. Do projektu je zakomponována uložením odpovídajícího souboru do adresáře *libs*.

```
dependencies {
        ...
        implementation files('../libs/commons-net-3.8.0.jar')
        ...
}
```

Generování síťového provozu lze v režii protokolu FTP řešit stahováním či nahráváním souborů na daný server. V rámci aplikace je pro demonstraci tohoto generování využito nahrávání předem daného souboru (o velikosti cca 4 MB) na server. Před samotným přenosem souboru je nicméně nutná autentizace uživatele.

```
private FTPClient client = null;
private final String host, username, password, assetsFileName;
private final FtpResponse ftpResp;
private int port, totSize, progress;
...

client = new FTPClient();
try {
    client.connect(host, port);
    client.login(username, password);
} catch (IOException e) {
    e.printStackTrace();
    Log.d(TAG, "Error: Can not connect to FTP server");
}
```

Pokud jsou přihlašovací údaje validní, vytvoří se uživateli při prvním přihlášení na základě výše zmíněného konfiguračního souboru *ldap.conf* na FTP serveru *Proftpd* vlastní účet a domovský adresář (např. */ftpshare/PETR.BLAZEK*), kterému je rovněž nastaveno vlastnictví. Do tohoto adresáře následně

uživatel ukládá svá data a pracuje s nimi. Následující ukázka ze zdrojového kódu demonstruje proces nahrání souboru na server.

```
if(ftpUpload(assetsFileName, assetsFileName, context)) {
    ftpResp.ftpUploadFinished("FTP: File " + assetsFileName + " uploaded
                              successfully.");
} else {
    ftpResp.ftpUploadFinished("FTP: File " + assetsFileName + " was not
                              uploaded.");
}
```

Pokud je výstupem metody *ftpUpload* hodnota *true*, pomocí instance rozhraní *FtpResponse* je uživateli po dokončení procesu doručena na obrazovku zpráva o výsledku transferu souboru.

Parametry metody *ftpUpload* jsou název zdrojového souboru v úložišti zařízení, dále název výstupního souboru (tzn. jak bude soubor pojmenován po uložení na FTP server) a rovněž instance třídy *Context* z důvodu přístupu k požadovanému adresáři, ve kterém se zdrojový soubor nachází.

Se zdrojovým souborem pracuje třída *FileInputStream*, díky které je možné zjistit celkovou velikost souboru a dále třída *CopyStreamAdapter*, jež je součástí knihovny *Apache Commons.* Tato třída implementuje metodu *bytesTransferred*, díky které je uživateli na obrazovce zobrazován aktuální stav přeneseného počtu bytů daného souboru. O publikování tohoto stavu se stará metoda *publishProgress*, která je součástí *AsyncTask* a při jejím volání se volá metoda *onProgressUpdate*, která tento stav přenáší uživateli na *UIthread* (hlavní vlákno). Samotný proces nahrání souboru pak řeší metoda *storeFile*, která se volá na instanci třídy *FTPClient*.

```
public boolean ftpUpload(String srcFileName, String desFileName, Context context)
{
    boolean status = false;
    try {
        AssetManager am = context.getApplicationContext().getAssets();
        AssetFileDescriptor afd = am.openFd(srcFileName);
        FileInputStream srcFileStream = afd.createInputStream();
        totalSize = srcFileStream.available() / 1024;

        CopyStreamAdapter streamListener = new CopyStreamAdapter() {
            @Override
            public void bytesTransferred(long totBytes, int bytes, long strS) {
                progress = (int)(totBytes / 1024);
                publishProgress(progress);
            }
        };

        client.setCopyStreamListener(streamListener);
        status = client.storeFile(desFileName, srcFileStream);
        srcFileStream.close();
        return status;
    }
    catch (Exception e) {
        Log.d(TAG, "FTP: Upload failed");
        e.printStackTrace();
    }
    return status;
}

@Override
protected void onProgressUpdate(Integer... values) {
    super.onProgressUpdate(values);
    ftpResp.kbTransferred(MessageFormat.format("{0} / {1} kB",progress, totSize));
}
```

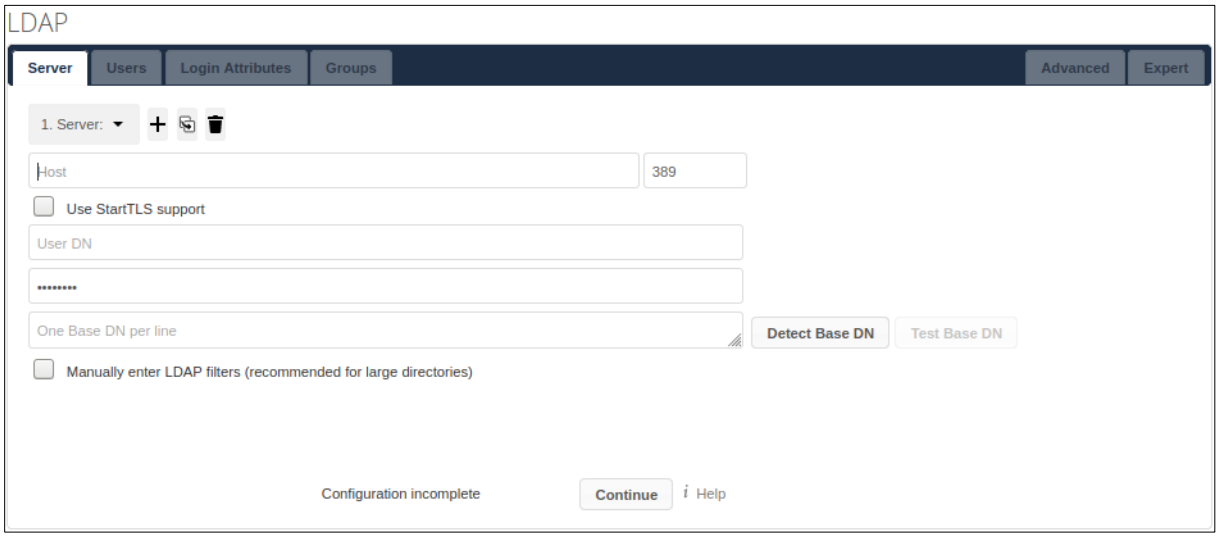
4.6.5.3 *ownCloud*

Pozn.: Generování síťového provozu pro protokol WebDAV, který využívá služba ownCloud, nebylo součástí zadání této práce. Z pohledu ověřování uživatelů vůči LDAP se však jeví jako zajímavá alternativa k protokolu FTP.

Služba *ownCloud* je open source řešením pro ukládání dat v cloudu. Jedná se o alternativní multiplatformní službu ke známějším úložištím jako jsou *Google Drive* či *Dropbox*. Její architektura je založena na dvojici klient – server, kde uživatelé přistupují k datům přes webový prohlížeč nebo pomocí klientů, kteří existují jednak pro mobilní zařízení a jednak pro desktopy. Služba *ownCloud* je založena na protokolu WebDAV, který rozšiřuje protokol HTTP a umožňuje tak přes webový server pracovat se soubory, čímž odpadá nutnost pracovat s protokoly jako FTP, NFS či Samba.

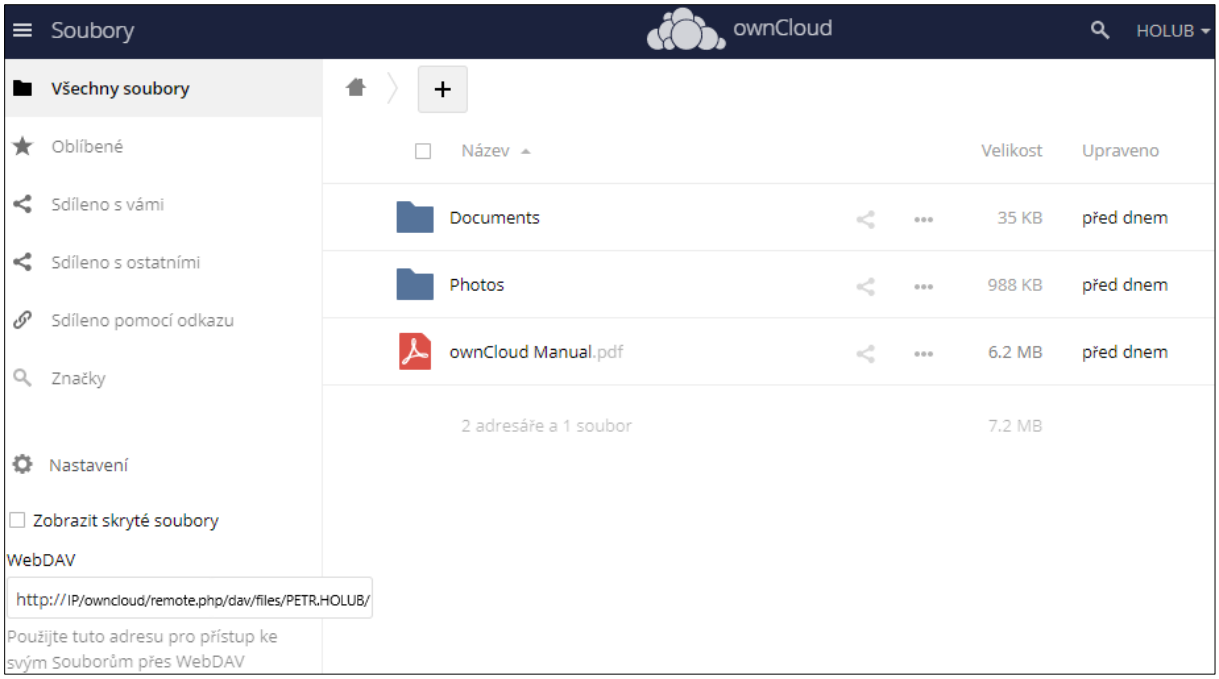
[46]

Po instalaci této služby na unixový systém je nutné nakonfigurovat ověřování přihlašovacích údajů vůči AD. Toho je možné docílit prostřednictvím webového portálu, ve kterém má administrátorský účet možnost nastavit autentizaci uživatelů (viz obrázek 4.9). Jelikož se v této práci řeší nešifrovaný proces autentizace jednotlivých uživatelů, není proto nutné importovat certifikát LDAP serveru na *ownCloud* server, který by eventuální šifrování zajišťoval.



Obrázek 4.9: *Nastavení autentizace LDAP uživatelů vůči ownCloud serveru*

Po vyplnění polí v jednotlivých záložkách je ověřování uživatelů vůči AD nastaveno a uživatel tak má možnost vyzkoušet, zda se může ke službě přihlásit s vlastními přihlašovacími údaji. Z hlediska klientské části, která je implementována do aplikace v OS *Android*, je nutné zjistit WebDAV adresu uživatele, která mu zajišťuje přístup k souborům. Ta je každému uživateli zveřejněna na obrazovce po přihlášení se přes webový portál (vlevo dole na obrázku 4.10).



Obrázek 4.10: *Webový portál služby ownCloud pro přihlášeného uživatele*

Službu *ownCloud* lze ve vývojovém prostředí *Android Studio* implementovat použitím knihovny *Sardine.* Do projektu lze naimportovat následovně:

```
dependencies {
        ...
        implementation 'com.github.thegrizzlylabs:sardine-android:v0.6'
        ...
}
```

Proces nahrání souboru je obdobný jako u FTP serveru, kdy nejprve je nutné se ke službě přihlásit. Pokud je proces autentizace úspěšný, využije se třídy *InputStream* pro získání cesty k požadovanému adresáři, ve kterém se nachází zdrojový soubor. Metoda *put* slouží pro nahrání souboru do cloudového úložiště. Třetí parametr této metody je však nutný poskytnout ve formě pole bytů, tudíž je nutné vytvořit dočasnou kopii souboru. O výsledku procesu nahrání souboru se uživatel dozví opět na základě instance třídy *FtpResponse*, která je volána po ukončení *AsyncTasku*, v rámci kterého celý tento proces proběhl.

```
@Override
protected String doInBackground(Void... voids) {

    Sardine sardine = new OkHttpSardine();
    sardine.setCredentials(username, password, true);

    String result = "OwnCloud: Error uploading file " + assetsFileName;
    try {
        InputStream is = context.getAssets().open(assetsFileName);
        File file = createFileFromInputStream(is);
        if (file != null && file.exists()) {
            byte[] data = FileUtils.readFileToByteArray(file);
            sardine.put(webDavAddress + assetsFileName, data);
            if(file.delete()) {
                Log.d(TAG, "OwnCloud: Temp file " + assetsFileName + " deleted");
            }
        }
        if(sardine.exists(webDavAddress + assetsFileName)) {
```

```
                result = "OwnCloud: File " +assetsFileName+ " uploaded successfully";
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return result;
}

@Override protected void onPostExecute(String ocResult) {
        super.onPostExecute(ocResult);
        ftpResponse.ownCloudUploadFinished(ocResult);
}
```

4.6.5.4 ***SIP***

Session Initiation Protocol je zodpovědný za přenos signalizace v internetové telefonii (VoIP). Stará se o multimediální komunikační relace, probíhající v reálném čase tak, že je zahajuje, modifikuje a ukončuje. Těmito relacemi se myslí hlasové hovory, zprávy, videohovory a další komunikační aplikace, které probíhají mezi dvěma koncovými body. Pro popis relací využívá protokolu SDP, kdežto samotný přenos hlasu či videa zajišťuje protokol RTP. SIP funguje na architektuře klient – server a standardně naslouchá na UDP portu 5060.

[47][48]

Co se týče konfigurace samotného SIP serveru, byla využita open source platforma, implementující telefonní ústřednu pro IP sítě, zvaná *Asterisk*. Jak již bylo zmíněno výše, protokol SIP je signalizační protokol a jeho úkolem není přenášet uživatelská data, nýbrž signalizační zprávy. Síťový provoz tak v režii SIP protokolu znamená posílání konkrétních SIP zpráv mezi klientem a serverem.

Vzhledem k charakteristice projektu, kde je nutné využít nešifrovaných přihlašovacích údajů, se v rámci protokolu SIP jeví jako možnost poskytnout tyto údaje pro registraci SIP účtu. *Asterisk* nicméně neumožňuje přenášet heslo SIP účtu v nešifrované podobě, neboť pro jeho přenos sítí se vždy využívá autentizační schéma *digest*. Toto schéma kombinuje dohromady požadavek ze strany serveru o autentizaci (tzv. *nonce*), uživatelské jméno, specifikaci SIP serveru (tzv. *realm*), heslo uživatele a URI požadavek a na základě této kombinace si klient vytvoří hashovací klíč pomocí hashovací metody *MD5*. Tento klíč nakonec pošle serveru pro registraci.

I přes výše zmíněné zabezpečení však lze nalézt způsob, jak heslo přenést v nešifrované formě pomocí protokolu SIP. Server sice tento požadavek ze strany klienta odmítne, nicméně samotné heslo se v nešifrované podobě přenese. Jedná se o způsob, kdy klient své heslo zadá omylem přímo do svého jedinečného SIP identifikátoru (tzv. *SIP URI*), které má následující schéma:

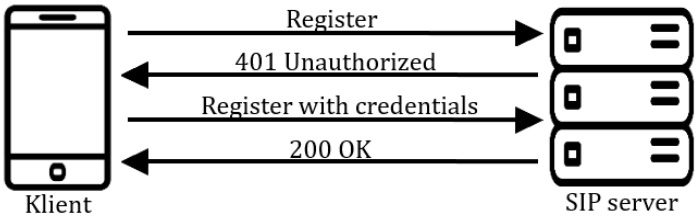
```
sip:user@domain
```

Po chybě, způsobené uživatelem, tak může SIP URI vypadat následovně:

```
sip:PETR.SOUKUPV4Mvh#42@1.2.3.4
```

Výše zmíněná chybná identifikace tak může v potencionálním útočníkovi například vzbudit dojem, že uživatel se ve své aplikaci zapomněl překliknout do daného pole, určeného heslu, a namísto toho jej zadal do kolonky společně s názvem uživatele.

Následující obrázek znázorňuje výměnu SIP zpráv mezi aplikací (klientem) a *Asteriskem* (SIP serverem) při registraci SIP účtu (koncového bodu/endpointu). Právě tato výměna zpráv je realizována v prostředí *Android Studio* v rámci síťového provozu pro protokol SIP.



Obrázek 4.11: *Výměna SIP zpráv mezi klientem a serverem při registraci SIP účtu*

Pod každým takovýmto endpointem si lze představit například mobilní koncová zařízení nebo softphone aplikace, kde každý endpoint umožňuje vytvářet a přijímat hovory skrze SIP server jakožto prostředníka. Jednotlivé endpointy však musí mít na SIP serveru své platné konfigurace, které jim definují jejich vlastnosti.

Integrace LDAP uživatelů se v *Asterisku* řeší pomocí modulu *res_config_ldap* a konfigurací příslušných souborů. V této práci však bylo využito alternativní metody této integrace. Jelikož se uživatelé na Active Directory vytvářejí pomocí skriptu (zmíněného v kapitole 4.5.2), lze tento skript využít zároveň pro vytvoření koncového bodu na SIP serveru. Výňatek z tohoto skriptu, řešící tuto problematiku, je znázorněn na následujícím obrázku.

```
#Asterisk
$string = "[$($Attributes.SamAccountName)]$'\n'
          type=endpoint$'\n'
          context=from-internal$'\n'
          allow=!all,alaw,g722$'\n'
          aors=$($Attributes.SamAccountName)$'\n'
          auth=$($Attributes.SamAccountName)$'\n'$'\n'
          [$($Attributes.SamAccountName)]$'\n'
          type=aor$'\n'
          max_contacts=1000$'\n'$'\n'
          [$($Attributes.SamAccountName)]$'\n'
          type=auth$'\n'
          auth_type=userpass$'\n'
          username=$($Attributes.SamAccountName)$'\n'
          password=$($password)"
$endpoint = $string -replace '\s',''
$sam = $($Attributes.SamAccountName)
$sam = $sam -replace '\s',''

plink -ssh [redacted] -pw [redacted] "echo $($endpoint) > /etc/asterisk/ad_endpoints/$($sam).conf"
plink -ssh [redacted] -pw [redacted] service asterisk reload
```

Obrázek 4.12: *Část PowerShell skriptu, vytvářející koncový bod na SIP serveru*

Samotná data, která je potřebná vložit do konfiguračního souboru na SIP serveru, jsou rozdělena do tří sekcí – *endpoint*, *aor* a *auth*, kde název těchto sekcí je definován mezi ostrými závorkami. Sekce *endpoint* poskytuje koncovému bodu základní funkcionalitu a obsahuje vazby na další sekce. Sekce *aor* zase informuje *Asterisk*, jak lze koncový bod kontaktovat a nakonec sekce *auth* obsahuje přihlašovací údaje, které budou využity při registraci SIP účtu na server. Tato data jsou jako řetězec vepsána příkazem *echo* do souboru, který se na serveru vytvoří ve specifikovaném adresáři pod názvem uživatele (např. */etc/asterisk/ad_endpoints/PETR.BLAZEK.conf*). Jelikož se koncové body v *Asterisku* definují v souboru *pjsip.conf*, obsahuje tento soubor příkaz, kterým zajistí import všech nově vytvořených souborů z adresáře */etc/asterisk/ad_endpoints*. Po přidání uživatele do konkrétního souboru pak musí *Asterisk* znovu načíst svou konfiguraci, což zajistí příkaz *service asterisk reload*. Vytvoření spojení mezi *Active Directory* a SIP serverem je realizováno SSH tunelem pomocí nástroje *plink*.

Android poskytuje v rámci svého developmentu vlastní API rozhraní protokolu SIP. Díky tomu lze v aplikaci pro OS *Android* implementovat klientskou část tohoto protokolu bez nutnosti importovat externí knihovny či závislosti. Jediné, co je potřeba pro zpřístupnění tohoto API udělat, je v konkrétní třídě importovat následující balíček:

```
import android.net.sip.*;
```

Základní třídou, která zprostředkovává toto API rozhraní, je třída *SipManager*. Nejprve je nutné získat instanci této třídy. Následně je nutné sestavit tzv. SIP profil, který se skládá z přihlašovacích údajů pro registraci SIP účtu a IP adresy SIP serveru. Samotnou registraci tohoto účtu na serveru provádí metoda *register*, jejímiž parametry jsou již zmíněný SIP profil, doba platnosti registrace (v sekundách) a instance třídy *SipRegistrationListener*, která poskytuje zpětnou vazbu k jednotlivým událostem procesu registrace. Na základě výměny SIP zpráv mezi klientem a serverem při registraci SIP účtu (obrázek 4.11) se klientovi nejprve doručí zpráva s chybou (skrze metodu *onRegistrationFailed – 401 Unauthorized*), následovaná informacemi o probíhající registraci a nakonec o dokončení registrace.

```
private SipManager sipManager;
private SipProfile sipProfile;
private String sipMessage;
private final String server, user, pw;
private final SipResponse sipResponse;
...

sipManager = SipManager.newInstance(context);

try {
    SipProfile.Builder sB = new SipProfile.Builder(user,server).setPassword(pw);
    sipProfile = sB.build();
    if(sipProfile != null) {
        sipManager.open(sipProfile);

        SipRegistrationListener srl = new SipRegistrationListener() {
            @Override
            public void onRegistering(String localProfileUri) {
                sipMessage = "Registering SIP account " + user;
                publishProgress(sipMessage);
            }
            @Override
            public void onRegistrationDone(String localProfUri, long expiryTime) {
                sipMessage = "SIP account "+user+" registered on server "+server;
                publishProgress(sipMessage);
                sipManager.close(sipProfile.getUriString());
                sipResponse.sipFinished(sipMessage);
            }
            @Override
            public void onRegistrationFailed(String lpu, int eCode, String eMsg) {
                sipMessage = "SIP registration failed. ErrorCode: " + eCode +
                             ", ErrorMessage: " + eMsg;
                publishProgress(sipMessage);
            }
        };

        sipManager.register(sipProfile, 30, srl);
        sipManager.setRegistrationListener(sipProfile.getUriString(), srl);
    }
} catch (SipException e) {
    e.printStackTrace();
}
```

4.6.5.5 ***IMAP / POP3***

Internet Message Access Protocol je poštovní protokol, který se využívá pro přístup k e-mailové schránce na vzdáleném webovém serveru z místního e-mailového klienta. Umožňuje přístup ke konkrétnímu e-mailovému účtu z více klientů zároveň, díky čemuž je vhodnou volbou pro uživatele, kteří přistupují ke svému e-mailovému účtu z různých míst nebo jej mezi sebou sdílejí. E-maily na serveru zůstávají a uživatel má možnost s nimi pracovat. Protokol IMAP využívá port 143 pro nešifrovanou a port 993 pro šifrovanou komunikaci.

Post Office Protocol (verze 3) se naproti tomu využívá k přijímání e-mailů ze vzdáleného serveru do místního e-mailového klienta. Tento protokol umožňuje stahovat zprávy do zařízení, čímž je umožňuje uživateli číst i v offline režimu. Ten si může nastavit, zda jsou e-maily po stažení z POP3 serveru odebrány nebo na něm zůstanou. Výhoda použití tohoto protokolu spočívá kromě již zmíněné offline dostupnosti e-mailových zpráv také v redukci místa, které zabírá e-mailový učet na serveru. Naproti tomu nevýhodou POP3 je situace, kdy uživatel přistupuje ke svému e-mailovému účtu z více míst (zařízení). Protokol POP3 využívá port 110 pro nešifrovanou a port 995 pro šifrovanou komunikaci. [49]

V této práci byl využit IMAP/POP3 server, zvaný *Dovecot.* Jedná se o open source poštovní server pro unixové systémy, který jednak umožňuje ověřovat identitu uživatelů z adresářového LDAP serveru a rovněž podporuje autentizaci uživatelů mechanismem PLAIN, díky kterému lze přenášet přihlašovací údaje po síti v nešifrované podobě.

Autentizace LDAP uživatelů se zde řeší úpravou souboru */etc/dovecot/dovecot-ldap.conf*, do kterého je nutné uvést následující informace, potřebné k navázání spojení s *Active Directory* serverem:

```
hosts               = IP adresa:port
ldap_version        = 3
auth_bind           = yes
dn                  = Uživatel@AD_doména
dnpass              = Heslo
base                = ou=services,dc=AD_doména
scope               = subtree
deref               = never
user_filter         = (&(userPrincipalName=%u)(objectClass=person)
                      (!(userAccountControl:1.2.840.113556.1.4.803:=2)))
pass_filter         = (&(userPrincipalName=%u)(objectClass=person)
                      (!(userAccountControl:1.2.840.113556.1.4.803:=2)))
pass_attrs          = userPassword=password
default_pass_scheme = PLAIN
user_attrs          = =home=/var/vmail/vmail1/%Ld/%Ln/Maildir/,
                      =mail=maildir:/var/vmail/vmail1/%Ld/%Ln/Maildir/
```

Obsah tohoto souboru je následně nutné importovat do konfiguračního souboru */etc/dovecot/dovecot.conf*, který obsahuje veškerá globální nastavení poštovního serveru.

Po konfiguraci serverové části pro protokoly IMAP a POP3 bylo nutné v aplikaci naimplementovat rovněž odpovídající klientskou část. Toho bylo docíleno s využitím lightweight verze *JavaMail API* [51], kterou lze do aplikace naimportovat následovně:

```
dependencies {
       implementation files('../libs/additionnal.jar')
       implementation files('../libs/activation.jar')
       implementation files('../libs/mail.jar')
}
```

Důvod využití této lightweight verze *JavaMail API* spočívá v její jednoduchosti. Její využití je pro IMAP a POP3 téměř identické, kde rozdíl je víceméně pouze ve způsobu manipulace s obsahem schránky. Následující výňatek ze zdrojového kódu je pro protokol IMAP, nicméně pro protokol POP3 stačí pouze upravit obsah textových řetězců („imap“ zaměnit s „pop3“, respektive „IMAP“ s „POP3“).

Do instance třídy *Properties* (dostupné z balíčku *java.util*) je nejprve nutné vložit odpovídající vlastnosti, které musí klient splňovat, aby mu poštovní server umožnil vzájemnou komunikaci. Na základě těchto vlastností je dále vytvořena relace (třída *Session*).

```
Properties p = new Properties();
p.put("mail.imap.host", server);
p.put("mail.imap.user", email);
p.put("mail.imap.port", port);
p.put("mail.imap.auth.mechanisms", "PLAIN");

Session session = Session.getInstance(p);
String result = "IMAP server: Chyba připojení";
try {
    Store store = session.getStore("imap");
    store.connect(server, email, password);

    if(store.isConnected()) {
        Log.d(TAG, "Connected to IMAP server");
        Folder in = store.getFolder("INBOX");
        in.open(Folder.READ_ONLY);
```

Na instanci třídy *session* se následně volá metoda *getStore*, díky které aplikace získá odkaz na samotnou e-mailovou schránku. K této schránce se dále autentizuje pomocí přihlašovacích údajů uživatele. Pokud je autentizace úspěšná, využije se třída *Folder*, jejíž instance získá přístup k adresářové struktuře e-mailové schránky.

Nyní nastává bod, kde se implementace IMAP liší od POP3. Pro IMAP může aplikace například vyhledat všechny nové e-maily a vypsat jejich počet uživateli na obrazovku nebo zobrazit jejich čas doručení a předmět. Protokol POP3 naproti tomu prohledá schránku a jednotlivé zprávy uloží do lokálního úložiště v zařízení.

```
//IMAP
        Message[] ms = in.search(new FlagTerm(new Flags(Flags.Flag.SEEN), false));
        result = "IMAP server: " + ms.length + " nových e-mailů.";
        for ( Message m : ms ) {
            Toast.makeText(c, "Date: " + m.getSentDate() + " Subject:"
                                + m.getSubject(), Toast.LENGTH_SHORT).show();
        }
```

```
//POP3
        Message[] ms = in.getMessages();
        result = "POP3 server: " + ms.length + " e-mailů ke stažení.";
        if(ms.length != 0) {
            for (Message m : ms) {
                File f = new File(c.getFilesDir(), m.getFileName());
                OutputStream os = new FileOutputStream(f);
                m.writeTo(os);
            }
        }
```

4.6.5.6 ***SMTP***

Poslední protokol, pro který bylo v této práci umožněno generování síťového provozu je *Simple Mail Transfer Protocol.* Jak již vyplývá z jeho názvu, stará se o odesílání e-mailů skrze internet – tedy o odchozí poštu. Standardně využívá port 25 pro nešifrovanou a port 465 pro šifrovanou komunikaci. Využívá rovněž port 587, pro který je možnost šifrování volitelná.

SMTP server, jehož konfigurace byla součástí této práce, nese označení *Postfix*. Jedná se o open source software, určený pro unixové systémy. Samotný *Postfix*, jakožto server pro odchozí poštu, je úzce spjatý s nástrojem *Dovecot*, který slouží jako server pro poštu příchozí.

Při každém odeslání e-mailu klientem se musí SMTP server rozhodnout, zda je klient oprávněn tuto poštu odesílat. Pokud klient pochází ze stejné lokální sítě jako SMTP server, je mu toto oprávnění obvykle uděleno. Klienti, kteří pochází z neznámých sítí, proto musí vůči serveru provádět proces autentizace. K tomuto procesu slouží tzv. SASL autentizace (definována v RFC 2222). *Postfix* tuto autentizaci přímo neimplementuje, nicméně využívá externích autentizačních backendů, mezi které patří například PAM, SQL databáze, LDAP nebo samotný IMAP server. Další možností, jak ověřit identitu klienta z neznámé sítě, je manuální přidání IP adresy této sítě do konfiguračního souboru */etc/postfix/main.cf* pod položku *mynetworks*. SASL se rovněž využívá pro ověření, zda e-mailová adresa, ze které je e-mail odesílán, je platná, respektive odpovídá platnému LDAP uživateli. V souboru */etc/postfix/main.cf* proto musí být tito LDAP (AD) uživatelé namapováni ke správné proměnné. Tento soubor dále obsahuje základní nastavení, týkající se například šifrování, nastavení domény nebo omezení jak na straně odesílatele, tak příjemce.

[50]

Implementace SMTP klienta je v aplikaci řešena obdobným způsobem, jako v případě protokolů IMAP a POP3 – importem lightweight verze *JavaMail API*, o níž pojednává [51]. Co se týče autentizace, rozdíl v implementaci oproti IMAP/POP3 je minimální.

```
Properties p = new Properties();
p.put("mail.smtp.host", server);
p.put("mail.smtp.user", from);
p.put("mail.smtp.port", port);
p.put("mail.smtp.auth.mechanisms", "PLAIN");
p.put("mail.smtp.auth", "true");

Session session = Session.getInstance(p, new Authenticator() {
    @Override
    protected PasswordAuthentication getPasswordAuthentication() {
        return new PasswordAuthentication(from, password);
    }
});

String result = "SMTP server: Chyba připojení";
```

Pro odeslání e-mailu je nutné jej nejprve vytvořit, což umožňuje třída *MimeMessage*. Instanci této třídy lze přiřadit vlastnosti jako odesílatel, příjemce, předmět a samotný text zprávy. O poslání e-mailu se stará abstraktní třída *Transport* pomocí metody *send*.

```
try {
    Transport transport = session.getTransport("smtp");
    MimeMessage m = new MimeMessage(session);
    m.setFrom(new InternetAddress(from));
    m.addRecipient(Message.RecipientType.TO, new InternetAddress(to));
```

```
    m.setSubject("Předmět e-mailu");
    m.setText("Text");
    Transport.send(m);
    result = "SMTP server: E-mail pro " + to + " byl odeslán.";
    transport.close();
} catch (MessagingException e) {
    e.printStackTrace();
}
```

4.6.6 Proces generování síťového provozu

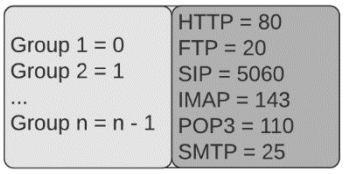
Tato kapitola pojednává o implementaci generování síťového provozu do aplikace jako celku. Zaměřuje se na propojení jednotlivých tříd, které mají na starosti generování síťového provozu pro jednotlivé protokoly společně s třídami, simulujícími chování běžného uživatele na internetu v podobě navštěvování různých webových serverů. Kapitola rovněž objasňuje chování dialogu, v rámci kterého celý proces generování probíhá.

Jak již bylo zmíněno na konci kapitoly 4.6.4, která pojednává o nastavení parametrů pro generování síťového provozu, kliknutím na tlačítko „Spustit generování“ se vyvolá dialog (třída *GeneratingTrafficDialog*), v jehož režii celý proces generování síťového provozu probíhá. Tento dialog je potomkem třídy *DialogFragment* a implementuje chování metod jednotlivých rozhraní, které jej informují o ukončení komunikací s jednotlivými servery.

Třída *GeneratingTrafficDialog* nejprve zjišťuje, jaké protokoly uživatel vybral. To se dozví díky objektu *Bundle*, v rámci kterého bylo do této třídy posláno pole hodnot pravda/nepravda. Kromě tohoto pole získá dialog dále pole IP adres pro jednotlivé servery, přihlašovací údaje, které bude využívat pro připojení se k jednotlivým serverům a rovněž minimální čas, po který bude proces generování probíhat (v případě, že se nevyužije možnost generování bez čekání).

Jelikož je předem známý počet skupin webových stránek (viz kapitola 4.6.3), které bude nutné navštívit a rovněž počet protokolů, se kterými se bude pracovat, je nutné navrhnout pořadí těchto skupin či protokolů, ve kterém budou síťový provoz generovat. Jednou z možností, jak toto provést, je následující teorie:

- Každé skupině webových stránek přidělit číslo od 0 do POČET_SKUPIN – 1 a každému protokolu přidělit číslo (např. dle portu protokolu). Čísla musí být unikátní.



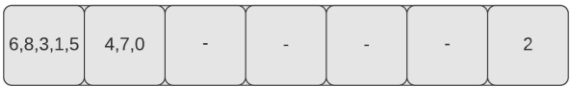
Obrázek 4.13: *Přidělení unikátních čísel skupinám webových stránek a protokolům*

- Čísla odpovídající jednotlivým skupinám webových stránek náhodně promíchat.

Groups	0, 1, 2, 3, 4, 5, 6, 7, 8
Order	6, 8, 3, 1, 5, 4, 7, 0, 2

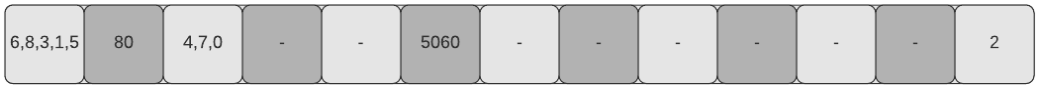
Obrázek 4.14: *Promíchání pořadí jednotlivých skupin webových stránek*

- Jelikož je protokolů celkem 6, vytvořit 7 polí (před a za každým protokolem) a každému z těchto sedmi polí přidělit náhodně skupiny webových stránek.



Obrázek 4.15: *Náhodné přiřazení skupin webových stránek do polí*

- Vytvořit pole o velikosti 13 (6 protokolů + 7 polí mezi nimi) a přiřadit každému sudému indexu tohoto pole čísla daných skupin webových stránek (začíná se indexem 0) a každému sudému indexu čísla, definující jednotlivé protokoly (vzor skupina1 - protokol1 - skupina2 - …)



Obrázek 4.16: *Sestavení výsledného pořadí, ve kterém se bude generovat síťový provoz pro webové stránky a vybrané protokoly HTTP a SIP*

Po vytvoření pořadí je dále nutné zahrnout do procesu generování síťového provozu také čas. Jak již bylo probíráno v kapitole 4.6.3, po každé návštěvě určité webové stránky následuje čekání, které se snaží napodobit chování uživatele, kdy na každé stránce nějakou chvíli pobývá, než přejde na další. Parametr, se kterým se tedy dále pracuje v rámci času, je průměrná doba čekání mezi jednotlivým voláním URL adres, kdy tato doba je vždy navýšena o čas, za který požadovaná webová stránka načte svůj obsah a vrátí ho do lokální proměnné v aplikaci. Každé webové stránce je pak rovněž přiřazena konstanta, určující, o kolik se má průměrná doba čekání snížit, respektive zvýšit. V případě komunikace s jednotlivými servery pro dané protokoly se vždy čeká na odpověď serveru, vyplývající z charakteristiky protokolu, což je řešeno právě pomocí rozhraní, které třída *GeneratingTrafficDialog* musí implementovat.

Třída *GeneratingTrafficDialog* dále zprostředkovává uživateli grafický výstup o probíhajícím generování síťového provozu, a to prostřednictvím obrazovky, která uživateli přehledně ukazuje uplynulý čas, právě prováděnou akci a taktéž aktuální postup v rámci generování provozu. Postup je kromě číselné reprezentace (aktuální postup / celkový postup) rovněž znázorněn pomocí komponenty *ProgressBar* v kruhové podobě, kdy s přibývajícím postupem se tato komponenta zaplňuje předem definovanou barvou.

Samotný proces generování však provádí třída *ClientMain*, která implementuje jak veškerou komunikaci s jednotlivými servery pomocí odpovídajících klientských částí, tak simulaci chování běžného uživatele navštěvováním webových stránek. Tato třída využívá abstraktní třídy *AsyncTask*, díky které je schopna pracovat na pozadí. V rámci programování s OS *Android* je rovněž nutné pracovat na pozadí při provádění síťových operací. Třída *GeneratingTrafficDialog* si s touto třídou vzájemně vyměňuje informace, kdy jí obecně definuje způsob, jak má generování probíhat (tzn. jaké protokoly a IP adresy využít, pořadí, práce s časem). Naproti tomu třída *ClientMain* poskytuje dialogu informace pomocí rozhraní, na jejichž základě jej informuje o aktuálním stavu procesu generování síťového provozu.

O navštěvování webových stránek a získání jejich HTML dokumentu pojednávala již kapitola 4.6.2 a o implementaci jednotlivých klientských částí pro vybrané protokoly kapitola 4.6.5. Třída *ClientMain* seskupuje tyto dvě témata do jednoho tak, že na základě společného pořadí jednotlivých skupin webových stránek nebo protokolů vybírá, která skupina či protokol bude v rámci procesu

generování síťového provozu následující. Pokud je vybrán konkrétní protokol, zavolá se klientská část, která má jeho implementaci na starost.

```
@Override
protected Void doInBackground(Void... voids) {
    for (int iInGroup = 0; iInGroup < currentGroup.size(); iInGroup++) {
        if(isCancelled()) {
            break;
        } else {
            if (iInGroup == 0)
                doc = new Document("empty");
            visitGroup(iInGroup);
        }
    }
    return null;
}

private void visitGroup(int iInGroup) {
    Random rn = new Random();
    switch(currentGroup.get(iInGroup)){
      case 80: {
        ClientHTTP http = new ClientHTTP(this, "http://" +protocolIps[0]+
                                                "/auth.php", username, password);
        http.execute();
        break;
      }
      case 20: {
        ClientFTP ftp = new ClientFTP(cont, this, this, protocolIps[1],
                                      username, password, 21, "wallpaper.png");
        ftp.setOwnCloud("http://" + ownCloudIp +
                        "/owncloud/remote.php/dav/files/"+ username + "/");
        ftp.execute();
        break;
      }

      ...

      case 0: {
        doc = website(doc, "https://www.seznam.cz", 0.2);
        doc = website(doc, doc.selectFirst(".article__text-box a")
                           .attr("abs:href"), 1.6);
        break;
      }

      ...
```

Proces generování síťového provozu je rovněž možné kdykoliv zastavit po kliknutí na tlačítko „Stop“, čímž se na instanci třídy *ClientMain* zavolá metoda *cancel*.

Proces generování síťového provozu na základě pořadí, které definuje obrázek 4.16 výše, by vypadal následovně (za předpokladu, že by uživatel zvolil délku trvání tohoto procesu generování na 1 minutu):

Po určení pořadí pro jednotlivé skupiny a protokoly se vypočítá průměrná doba čekání, kde 60 sekund je celková doba generování, stanovená uživatelem a 15 sekund je průměrný čas, během kterého se načte a uloží HTML dokument pro všech 30 webových stránek ze známého seznamu.

$$avgWaitTime = (60 - 15)/30 = 1.5\ sec$$

Průměrná doba čekání mezi jednotlivými webovými stránkami se tak rovná 1.5 sekund.

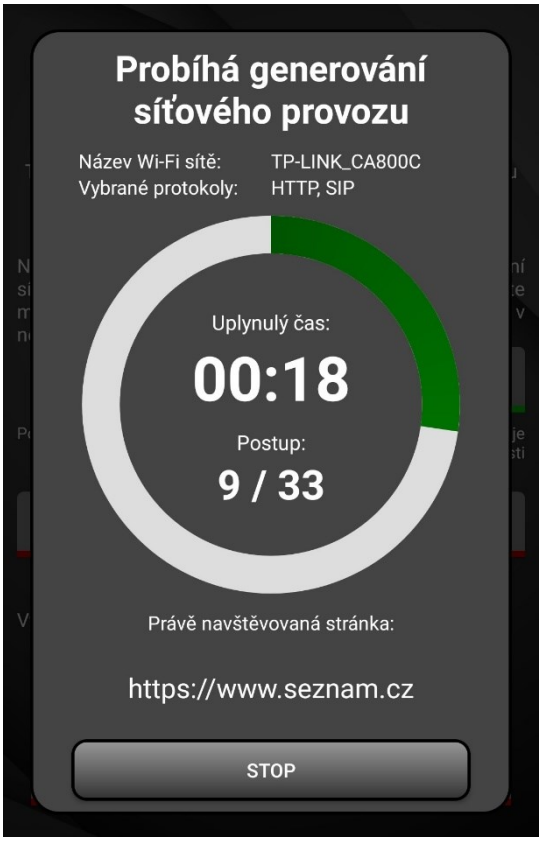
Nejprve aplikace navštíví všechny webové stránky, nacházející se v prvních pěti skupinách (očíslovaných 6, 8, 3, 1, 5), kde mezi jednotlivými webovými stránkami budou *AsyncTask* třídy, v rámci kterých je celý proces generování realizován, čekat průměrně 1.5 sekundy. Pokud bude mít stránka nastavena svou časovou konstantu např. na 2.0, bude čekat 3 sekundy, pokud na 0.1, bude čekat 150 milisekund.

Po navštívení pěti skupin webových stránek využije aplikace na základě stanoveného pořadí a využitých protokolů přihlašovacích údajů, poskytnutých z *Active Directory* tak, že je vloží do formuláře na webových stránkách HTTP serveru a následně potvrdí odeslání tohoto formuláře. Přenos těchto přihlašovacích údajů proběhl nešifrovaně, neboť HTTP server, který tyto webové stránky poskytuje, byl záměrně nakonfigurován, aby nepodporoval šifrování. Aplikace dále čeká, až z tohoto serveru dorazí odpověď (HTTP response).

Po získání této odpovědi se proces navštívení skupin webových stránek opakuje pro skupiny 4, 7 a 0. FTP protokol je další na řadě, avšak uživatelem nebyl vybrán, díky čemuž se přeskočí.

Na základě náhodného přidělení skupin webových stránek jednotlivým indexům pole je další v pořadí číslo 5060, náležící protokolu SIP, který byl v tomto případě vybrán kromě HTTP jako jediný další protokol. Spustí se tak proces registrace SIP účtu na SIP serveru, který byl popsán v kapitole 4.6.5.4 a platí stejný princip přenosu informací, jako u výše zmíněného HTTP.

Poslední skupině webových stránek, která ještě nebyla navštívena, bylo přiděleno číslo 2. S touto skupinou začne aplikace pracovat, jakmile dostane ze SIP serveru odpověď o úspěšné registraci SIP účtu.



Obrázek 4.17: *Dialog, zobrazující proces generování síťového provozu*

5 Vyhodnocení výsledků

Pokud se k síťovému provozu, který vytvořená aplikace vygenerovala v rámci bezdrátové sítě, dostane nepovolaná osoba a analyzuje jej, mezi daty, reprezentujícími aktivitu uživatele na různých webových stránkách vypozoruje i přenos nezabezpečených přihlašovacích údajů k jednotlivým serverům, jejichž odpovídající protokoly si uživatel v aplikaci vybral při generování síťového provozu. Pokud se rozhodne těchto přihlašovacích údajů zneužít a přihlásit se k těmto serverům sám, jeho aktivita bude vyzrazena, neboť jednotlivé servery mají nastaveno ověřování přihlašovacích údajů vůči *Active Directory*. Na základě informací, poskytnutých aplikací o bezdrátové síti, ve které se generování síťového provozu s konkrétními přihlašovacími údaji provedlo, je možné si na *Active Directory* zobrazit pomocí vhodného nástroje pro daného uživatele jeho historii přihlášení. Když se zjistí, že se daný uživatel přihlašoval v době, kdy neprobíhalo generování síťového provozu, bude se jednat o osobu, která těchto přihlašovacích údajů zneužila. Díky přidruženým informacím o bezdrátové síti je pak možné kontaktovat správce této sítě a upozornit ho na výskyt osoby, provádějící monitoring sítě.

Následující screenshoty z programu *Wireshark* demonstrují síťový provoz, který aplikace vygenerovala. Znázorňují nejprve komunikaci mezi aplikací a vybranými internetovými portály a dále se zaměřují na síťový provoz, který proběhl v rámci protokolů HTTP, FTP, SIP, IMAP, POP3 a SMTP. Ze screenshotů, zaměřujících se na konkrétní protokoly, lze vypozorovat zejména přenos nešifrovaných přihlašovacích údajů, které byly aplikací získány z LDAP serveru.

1052	18.250528	85.207.0.152	192.168.137.248	TLSv1.3	1506	Application Data [TCP segment of a reassembled PDU]
1062	18.253552	85.207.0.152	192.168.137.248	TLSv1.3	1506	Application Data [TCP segment of a reassembled PDU]
1080	18.256838	85.207.0.152	192.168.137.248	TLSv1.3	1506	Application Data [TCP segment of a reassembled PDU]
1095	18.260130	85.207.0.152	192.168.137.248	TLSv1.3	948	Application Data
1118	18.335804	192.168.137.248	77.75.77.164	TLSv1.3	90	Application Data
1121	18.352542	77.75.77.164	192.168.137.248	TLSv1.3	90	Application Data
1130	19.259386	77.75.74.172	192.168.137.248	TLSv1.3	90	Application Data
1153	21.384887	192.168.137.248	85.207.0.152	TLSv1.3	362	Application Data
1172	22.090080	85.207.0.152	192.168.137.248	TLSv1.3	1506	Application Data [TCP segment of a reassembled PDU]
1194	22.158339	85.207.0.152	192.168.137.248	TLSv1.3	1506	Application Data [TCP segment of a reassembled PDU]
1198	22.159553	85.207.0.152	192.168.137.248	TLSv1.3	1259	Application Data
1241	25.435636	192.168.137.248	95.129.100.15	TLSv1.2	583	Client Hello
1245	25.460629	95.129.100.15	192.168.137.248	TLSv1.2	758	Server Hello, Certificate, Server Key Exchange, Server Hello
1250	25.570973	192.168.137.248	95.129.100.15	TLSv1.2	216	Client Key Exchange, Change Cipher Spec, Encrypted Handshake
1251	25.599757	95.129.100.15	192.168.137.248	TLSv1.2	141	Change Cipher Spec, Encrypted Handshake Message
1253	25.610572	192.168.137.248	95.129.100.15	TLSv1.2	327	Application Data
1254	25.646945	95.129.100.15	192.168.137.248	TLSv1.2	471	Application Data
1259	25.725227	192.168.137.248	95.129.100.15	TLSv1.2	583	Client Hello
1260	25.746963	95.129.100.15	192.168.137.248	TLSv1.2	235	Server Hello, Change Cipher Spec, Encrypted Handshake Messag
1262	25.751591	192.168.137.248	95.129.100.15	TLSv1.2	141	Change Cipher Spec, Encrypted Handshake Message
1264	25.798650	192.168.137.248	95.129.100.15	TLSv1.2	359	Application Data
1278	25.873734	95.129.100.15	192.168.137.248	TLSv1.2	1506	Application Data [TCP segment of a reassembled PDU]
1286	25.900020	95.129.100.15	192.168.137.248	TLSv1.2	172	Application Data
1291	25.952930	192.168.137.248	77.75.76.63	TLSv1.3	90	Application Data
1314	28.786971	192.168.137.248	95.129.100.15	TLSv1.2	439	Application Data
1327	28.963134	95.129.100.15	192.168.137.248	TLSv1.2	1506	Application Data [TCP segment of a reassembled PDU]

Obrázek 5.1: *Komunikace mezi aplikací a vybranými webovými servery*

HTTP

	Time	Source	Destination	Protocol	Length	Info
41	5.023242	192.168.137.248	[redacted]	HTTP	565	POST /auth.php HTTP/1.1 (application/x-www-form-urlencoded)

```
HTML Form URL Encoded: application/x-www-form-urlencoded
  Form item: "username" = "PETR.STEPANEK"
    Key: username
    Value: PETR.STEPANEK
  Form item: "password" = "J1Hej/14"
    Key: password
    Value: J1Hej/14
  Form item: "sub" = "Login"
    Key: sub
    Value: Login
```

Obrázek 5.2: *Přenos nešifrovaných přihlašovacích údajů prostřednictvím protokolu HTTP*

FTP

	Time	Source	Destination	Protocol	Length	Info
2002	45.000746	[redacted]	192.168.137.248	FTP	105	Response: 220 ProFTPD 1.3.5 Server (Debian) [[redacted]]
2004	45.068390	192.168.137.248	[redacted]	FTP	74	Request: USER PETR.STEPANEK
2010	45.121052	[redacted]	192.168.137.248	FTP	95	Response: 331 Password required for PETR.STEPANEK
2011	45.127532	192.168.137.248	[redacted]	FTP	69	Request: PASS J1Hej/14
2015	45.589786	[redacted]	192.168.137.248	FTP	88	Response: 230 User PETR.STEPANEK logged in
2016	45.682924	192.168.137.248	[redacted]	FTP	82	Request: PORT 192,168,137,248,145,9
2020	45.736199	[redacted]	192.168.137.248	FTP	83	Response: 200 PORT command successful
2021	45.740083	192.168.137.248	[redacted]	FTP	79	Request: STOR RazerWallpaper.png
2026	45.903251	[redacted]	192.168.137.248	FTP	117	Response: 150 Opening ASCII mode data connection for RazerWa
7493	52.888528	[redacted]	192.168.137.248	FTP	77	Response: 226 Transfer complete
7496	52.953313	192.168.137.248	[redacted]	FTP	60	Request: QUIT
7504	53.005852	[redacted]	192.168.137.248	FTP	68	Response: 221 Goodbye.

Obrázek 5.3: *Generování síťového provozu pro protokol FTP*

SIP

14653	149.813323	192.168.137.217	[redacted]	SIP	518	Request: REGISTER sip:[redacted] (1 binding) |
14654	149.844499	192.168.137.217	[redacted]	SIP	495	Request: REGISTER sip:[redacted] (1 binding) |
14655	149.871077	[redacted]	192.168.137.217	SIP	632	Status: 401 Unauthorized |
14656	149.893503	[redacted]	192.168.137.217	SIP	617	Status: 401 Unauthorized |
14657	149.903358	192.168.137.217	[redacted]	SIP	763	Request: REGISTER sip:[redacted]:5060 (1 binding) |
14658	149.952374	[redacted]	192.168.137.217	SIP	587	Status: 200 OK (1 binding) |

```
Frame 14653: 518 bytes on wire (4144 bits), 518 bytes captured (4144 bits) on interface \Device\NPF_{CF433583-F637-4CFB-A1C2-23684549A05B}, id 0
Ethernet II, Src: XiaomiCo_17:8a:80 (e0:dc:ff:17:8a:80), Dst: e6:f8:9c:a7:5a:fd (e6:f8:9c:a7:5a:fd)
Internet Protocol Version 4, Src: 192.168.137.217, Dst: [redacted]
User Datagram Protocol, Src Port: 45761, Dst Port: 5060
Session Initiation Protocol (REGISTER)
▾ Request-Line: REGISTER sip:[redacted] SIP/2.0
    Method: REGISTER
  › Request-URI: sip:[redacted]
    [Resent Packet: False]
▾ Message Header
    Call-ID: b777ccbb8cf9e61552e2679c4c6252b3@192.168.137.217
    [Generated Call-ID: b777ccbb8cf9e61552e2679c4c6252b3@192.168.137.217]
  › CSeq: 2939 REGISTER
  › From: <sip:PETR.MORAVECU0Mof-21@[redacted]>;tag=861174810
  › To: <sip:PETR.MORAVECU0Mof-21@[redacted]>
  › Via: SIP/2.0/UDP 192.168.137.217:45761;branch=z9hG4bK6374bc9a9f078543b63e0cd653a2bc67313533;rport
    Max-Forwards: 70
    User-Agent: SIPAUA/0.1.001
  › Contact: <sip:PETR.MORAVECU0Mof-21@192.168.137.217:45761;transport=udp>
    Expires: 30
```

Obrázek 5.4: *Generování síťového provozu pro protokol SIP*

IMAP

	Time	Source	Destination	Protocol	Length	Info
7380	86.251730	[redacted]	192.168.137.238	IMAP	178	Response: * OK [CAPABILITY IMAP4rev1 SASL-IR LOGIN-REFERRALS ID ENABLE IDLE LI
7382	86.255722	192.168.137.238	[redacted]	IMAP	81	Request: A0 CAPABILITY
7388	86.278500	[redacted]	192.168.137.238	IMAP	222	Response: A0 OK Pre-login capabilities listed, post-login capabilities have mo
7389	86.284104	192.168.137.238	[redacted]	IMAP	89	Request: A1 AUTHENTICATE PLAIN
7391	86.307904	[redacted]	192.168.137.238	IMAP	70	Response: +
7392	86.311036	192.168.137.238	[redacted]	IMAP	136	Request: cGV0ci5zZXZjaWtA[redacted]uY3oAcGV0ci5zZXZjaWtA[redacted]uY3oATTlDaGEpMDY=
7394	86.353112	[redacted]	192.168.137.238	IMAP	477	Response: A1 OK Logged in
7395	86.357663	192.168.137.238	[redacted]	IMAP	81	Request: A2 CAPABILITY
7396	86.380150	[redacted]	192.168.137.238	IMAP	510	Response: A2 OK Capability completed (0.001 + 0.000 secs).
7397	86.395367	192.168.137.238	[redacted]	IMAP	75	Request: A3 NOOP
7398	86.417069	[redacted]	192.168.137.238	IMAP	110	Response: A3 OK NOOP completed (0.001 + 0.000 secs).
7399	86.450867	192.168.137.238	[redacted]	IMAP	84	Request: A4 EXAMINE INBOX
7400	86.482402	[redacted]	192.168.137.238	IMAP	333	Response: A4 OK [READ-ONLY] Examine completed (0.010 + 0.000 + 0.009 secs).
7401	86.508139	192.168.137.238	[redacted]	IMAP	88	Request: A5 SEARCH UNSEEN ALL
7402	86.531187	[redacted]	192.168.137.238	IMAP	122	Response: A5 OK Search completed (0.001 + 0.000 secs).
7403	86.537433	192.168.137.238	[redacted]	IMAP	76	Request: A6 CLOSE
7404	86.559912	[redacted]	192.168.137.238	IMAP	111	Response: A6 OK Close completed (0.001 + 0.000 secs).
7405	86.564194	192.168.137.238	[redacted]	IMAP	77	Request: A7 LOGOUT
7406	86.586056	[redacted]	192.168.137.238	IMAP	131	Response: A7 OK Logout completed (0.001 + 0.000 secs).

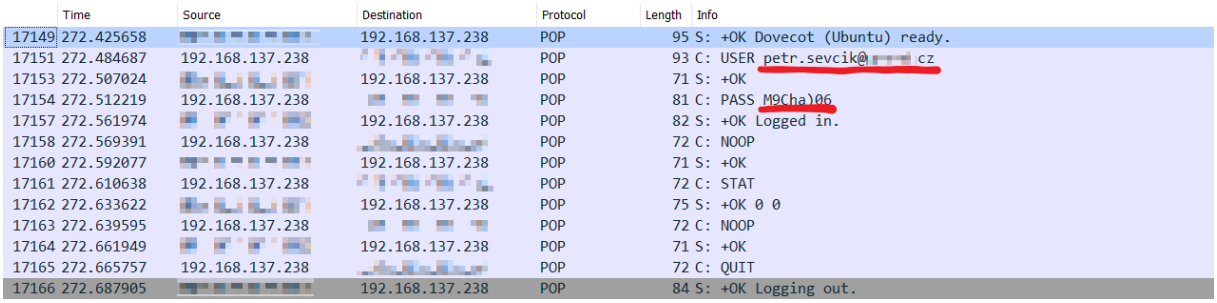
```
Frame 7392: 136 bytes on wire (1088 bits), 136 bytes captured (1088 bits) on interface \Device\NPF_{CF433583-F637-4CFB-A1C2-23684549A05B}, id 0
Ethernet II, Src: XiaomiCo_17:8a:80 (e0:dc:ff:17:8a:80), Dst: e6:f8:9c:a7:5a:fd (e6:f8:9c:a7:5a:fd)
Internet Protocol Version 4, Src: 192.168.137.238, Dst: [redacted]
Transmission Control Protocol, Src Port: 48604, Dst Port: 143, Seq: 39, Ack: 273, Len: 70
Internet Message Access Protocol
▾ Line: cGV0ci5zZXZjaWtA[redacted]uY3oAcGV0ci5zZXZjaWtA[redacted]uY3oATTlDaGEpMDY=\r\n
    Request: cGV0ci5zZXZjaWtA[redacted]uY3oAcGV0ci5zZXZjaWtA[redacted]uY3oATTlDaGEpMDY=
    Request Tag: cGV0ci5zZXZjaWtA[redacted]uY3oAcGV0ci5zZXZjaWtA[redacted]uY3oATTlDaGEpMDY=
```

Obrázek 5.5: *Generování síťového provozu pro protokol IMAP*

V tomto případě jsou přihlašovací údaje přenášeny sítí pomocí kódování *Base64*, které vždy kóduje 3 bajty (24 bitů) a převádí je do čtyř skupin po šesti bitech. Každá tato šestibitová hodnota je

převedena na číslo, ze kterého se odvozuje konečný *ASCII* znak. Dekódování této posloupnosti je primitivní i pro nezkušené uživatele, neboť na internetu lze najít mnoho nástrojů, které si s tímto kódováním poradí.

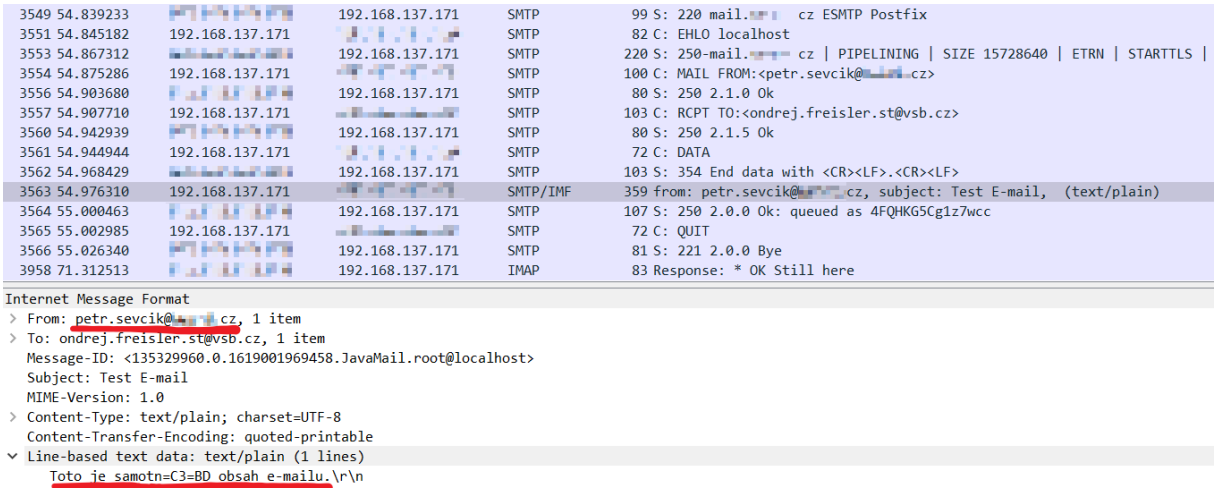
POP3



	Time	Source	Destination	Protocol	Length	Info
17149	272.425658	[redacted]	192.168.137.238	POP	95	S: +OK Dovecot (Ubuntu) ready.
17151	272.484687	192.168.137.238	[redacted]	POP	93	C: USER petr.sevcik@[redacted]cz
17153	272.507024	[redacted]	192.168.137.238	POP	71	S: +OK
17154	272.512219	192.168.137.238	[redacted]	POP	81	C: PASS M9Cha)06
17157	272.561974	[redacted]	192.168.137.238	POP	82	S: +OK Logged in.
17158	272.569391	192.168.137.238	[redacted]	POP	72	C: NOOP
17160	272.592077	[redacted]	192.168.137.238	POP	71	S: +OK
17161	272.610638	192.168.137.238	[redacted]	POP	72	C: STAT
17162	272.633622	[redacted]	192.168.137.238	POP	75	S: +OK 0 0
17163	272.639595	192.168.137.238	[redacted]	POP	72	C: NOOP
17164	272.661949	[redacted]	192.168.137.238	POP	71	S: +OK
17165	272.665757	192.168.137.238	[redacted]	POP	72	C: QUIT
17166	272.687905	[redacted]	192.168.137.238	POP	84	S: +OK Logging out.

Obrázek 5.6: *Generování síťového provozu pro protokol POP3*

SMTP



3549	54.839233	[redacted]	192.168.137.171	SMTP	99	S: 220 mail.[redacted] cz ESMTP Postfix
3551	54.845182	192.168.137.171	[redacted]	SMTP	82	C: EHLO localhost
3553	54.867312	[redacted]	192.168.137.171	SMTP	220	S: 250-mail.[redacted] cz | PIPELINING | SIZE 15728640 | ETRN | STARTTLS |
3554	54.875286	192.168.137.171	[redacted]	SMTP	100	C: MAIL FROM:<petr.sevcik@[redacted]cz>
3556	54.903680	[redacted]	192.168.137.171	SMTP	80	S: 250 2.1.0 Ok
3557	54.907710	192.168.137.171	[redacted]	SMTP	103	C: RCPT TO:<ondrej.freisler.st@vsb.cz>
3560	54.942939	[redacted]	192.168.137.171	SMTP	80	S: 250 2.1.5 Ok
3561	54.944944	192.168.137.171	[redacted]	SMTP	72	C: DATA
3562	54.968429	[redacted]	192.168.137.171	SMTP	103	S: 354 End data with <CR><LF>.<CR><LF>
3563	54.976310	192.168.137.171	[redacted]	SMTP/IMF	359	from: petr.sevcik@[redacted]cz, subject: Test E-mail, (text/plain)
3564	55.000463	[redacted]	192.168.137.171	SMTP	107	S: 250 2.0.0 Ok: queued as 4FQHKG5Cg1z7wcc
3565	55.002985	192.168.137.171	[redacted]	SMTP	72	C: QUIT
3566	55.026340	[redacted]	192.168.137.171	SMTP	81	S: 221 2.0.0 Bye
3958	71.312513	[redacted]	192.168.137.171	IMAP	83	Response: * OK Still here

```
Internet Message Format
> From: petr.sevcik@[redacted].cz, 1 item
> To: ondrej.freisler.st@vsb.cz, 1 item
  Message-ID: <135329960.0.1619001969458.JavaMail.root@localhost>
  Subject: Test E-mail
  MIME-Version: 1.0
> Content-Type: text/plain; charset=UTF-8
  Content-Transfer-Encoding: quoted-printable
v Line-based text data: text/plain (1 lines)
    Toto je samotn=C3=BD obsah e-mailu.\r\n
```

Obrázek 5.7: *Generování síťového provozu pro protokol SMTP*

Ověřování vůči LDAP bylo v rámci *Postfix* SMTP serveru nastaveno k ověření identity klienta – tedy zda je mu dovoleno odesílat poštu. Pro tento proces není uživatelské heslo sítí přenášeno. Naproti tomu lze zde zobrazit nešifrovaný obsah samotné e-mailové zprávy, kterou klient odeslal z aplikace.

6 Srovnání s jinými nástroji pro generování síťového provozu

Při srovnání vytvořené aplikace s již existujícími nástroji, umožňujícími generovat síťový provoz, je nutné brát zejména ohled na důvod jejich nasazení, respektive k čemu vlastně samotný proces generování síťového provozu slouží. Další klíčovou oblastí je pak platforma, pro kterou byly tyto nástroje navrženy.

Nástroje, generující síťový provoz, většinou slouží k zátěžovému testování v počítačových sítích. Na základě tohoto testování pak lze odhalit zranitelné nebo chybné oblasti síťové infrastruktury – tzv. *bottlenecky*, které mají negativní vliv na aktuální maximální přenosovou rychlost, kterou je síť schopna uživatelům zajistit. Jedná se de facto o diskrétní stav, ve kterém je tok dat omezen, neboť systém generuje větší objem dat, než je povolená kapacita sítě. Zátěžové testy mohou dále odhalit i potencionální útok hackerů, díky čemuž je možné provést preventivní kroky k podchycení této hrozby v samotném zárodku. Tyto nástroje obvykle napodobují chování skutečného síťového provozu a simulují zátěž, kterou je schopna daná síť zvládnout, aniž by byla ovlivněna kvalita služeb.

Těchto nástrojů, které generují síťový provoz kvůli testování zátěže, existuje celá řada, a to jak placených, tak těch zdarma. Mezi nejznámější open source nástroje v tomto směru patří následující programy:

[1][52]

6.1 Nping

Nping může být použit kromě generování síťového provozu pro širokou škálu protokolů rovněž jako nástroj *ping* pro detekci aktivních zařízení. Dále umožňuje provádět *ARP poisoning*, *DoS útoky* nebo provádět trasování paketů. Díky režimu *echo* mohu uživatelé pozorovat, jak se pakety při přenosu mezi zdrojovým a cílovým hostitelem mění a na základě této funkce mohou detekovat chybné pakety nebo nastavení firewallu. Nástroj nicméně není příliš uživatelsky přívětivý a podobně jako většina open source nástrojů pro generování síťového provozu není ideálním řešením pro firemní prostředí. Je dostupný pro platformy *Windows*, *macOS* i *Linux*.

[52]

6.2 Ostinato

Ostinato je všestranný nástroj, který bývá označován jako blízký společník nástroje *Wireshark*. Umožňuje vytvářet vlastní toky dat, které lze dále podrobně konfigurovat například s ohledem na rychlost přenosu či na základě objemu těchto dat. Dále umožňuje měnit obsah libovolného paketu a testovat tak případně vzniklé chyby. Při provádění zátěžových testů umožňuje uživateli zobrazit ztrátovost paketů díky možnosti vizualizace dat. Tento nástroj je rovněž vysoce kompatibilní a podporuje velkou většinu nejpoužívanějších síťových protokolů. Umožňuje rovněž práci s PCAP soubory. Je nicméně nutné poznamenat, že tento nástroj je bezestavový, což znamená, že nepodporuje stavová TCP připojení. Navíc není vhodný pro generování síťového provozu a jeho odesílání na webové stránky.

[52]

6.3 Packet Sender

Tento nástroj umožňuje provádět vlastní testování díky odesílání a přijímání paketů na porty, specifikované uživatelem. Co se týče způsobu použití, využívá tento nástroj jak příkazové řádky, tak grafického uživatelského rozhraní, což poskytuje flexibilitu. Jedná se o efektivní generátor a analyzátor síťového provozu, díky kterému lze najít například nedostatečné zabezpečení firewallu. Dále umožňuje testovat síťová API rozhraní pomocí integrovaných klientů nebo analyzovat malware na základě protokolů TCP, UDP nebo SSL. *Packet Sender* je kromě platforem *Windows*, *macOS* a *Linux* dostupný také pro mobilní zařízení skrze aplikaci *Google Play*, respektive *App Store*.

[52]

Smyslem aplikace, která byla v této práci vyvinuta, však není provádět zátěžové testování sítě. Generování síťového provozu v tomto případě nahrazuje chování běžného uživatele na internetu. V praxi pak toto generování provozu vypadá z hlediska monitoringu síťového provozu jako běžný síťový provoz, kdy během určitých logických intervalů navštěvuje uživatel různé webové servery. Aplikace slouží především jako tzv. *honeypot*, jejímž cílem je nalákat potencionální osobu, provádějící eventuální monitoring sítě, na odcizení nešifrovaných dat v podobě přihlašovacích údajů, které rovněž generuje. Tímto je definována hlavní podstata této aplikace, která je zcela odlišná ve srovnání s výše zmíněnými nástroji pro generování síťového provozu.

Závěr

Cílem této diplomové práce bylo vytvořit aplikaci pro operační systém Android, která umí generovat síťový provoz.

Dle bodů, které byly stanoveny v zadání této práce, se práce nejprve zaměřuje na problematiku síťového provozu z teoretického hlediska. Jelikož lze síťovým provozem označit de facto veškerou komunikaci, která na sítích probíhá, dalším probíraným tématem byly právě parametry sítí a rovněž referenční model OSI, neboť se jedná o témata úzce související s pojmem síťový provoz.

Operační systém Android představoval další stěžejní téma, které bylo nutné nejprve analyzovat v návaznosti na později vyvíjenou aplikaci. Součástí této analýzy byla taktéž kapitola, věnující se programování aplikací pro mobilní zařízení jako celku. V rámci této kapitoly bylo také nastíněno, že aplikace pro generování síťového provozu byla vyvíjena ve vývojovém prostředí Android Studio.

Po části práce, věnované teoretickým poznatkům, následovala část praktická, jejímž hlavním cílem byl právě návrh, vývoj a následná dokumentace již zmíněné aplikace. Možností, jak danou aplikaci navrhnout z praktického hlediska nicméně existovalo více, a to především díky tomu, že samotný proces generování síťového provozu není zcela jednoznačně definován. Klíčovou roli zde hrál rovněž účel tohoto generování, který blíže specifikoval oblast, ve které by se tento proces mohl uplatnit. V tomto případě se jednalo o bezpečnost bezdrátových sítí, neboť aplikace byla vyvíjena především kvůli tomu, aby přenášela sítí přihlašovací údaje pro předem dané protokoly (konkrétně HTTP, FTP, SIP, IMAP, POP3 a SMTP) prostřednictvím nešifrované komunikace. Samotné přihlašovací údaje získala aplikace z LDAP serveru (Microsoft Active Directory), do jehož sítě se dostala připojením skrze VPN a k těmto údajům následně přidala informace charakterizující bezdrátovou síť, ke které byla momentálně připojena. Na základě těchto údajů a informací se následně na LDAP serveru vytvořil uživatelský účet. Aby mohla být tato nešifrovaná komunikace realizovatelná, musel být pro každý z dříve uvedených protokolů rovněž nakonfigurován odpovídající server tak, aby jednak podporoval nešifrovanou komunikaci a zároveň ověřoval přihlašování uživatelů vůči LDAP serveru.

Tento nešifrovaný přenos dat v režii zmíněných protokolů byl klíčovou funkcí, kterou měla aplikace v rámci generování síťového provozu splňovat. Tato funkce totiž poskytuje možnost zjistit, zda byl síťový provoz bezdrátové sítě někým monitorován, a to právě na základě přenosu nešifrovaných přihlašovacích údajů touto sítí. Jakmile se totiž potencionální osoba, provádějící monitoring sítě, dostane k takovýmto údajům, existuje velká pravděpodobnost, že je zneužije a sama se pomocí nich na konkrétní servery přihlásí. Tímto však odhalí svou přítomnost v síti právě díky tomu, že servery mají nastaveno ověřování vůči LDAP serveru.

Při generovaní tohoto síťového provozu byl rovněž kladen důraz na napodobení chování běžného uživatele na internetu proto, aby přenos nezabezpečených přihlašovacích údajů nevzbuzoval podezření, že se jedná o automatizovaný proces, jehož cílem je pouze nalákat potencionálního útočníka na citlivá data. Jelikož se aplikace snaží napodobit toto chování generováním síťového provozu, nabízelo se mnoho postupů, jak toto chování interpretovat. Jednou z možností bylo vytvoření seznamu URL adres různých webových stránek. Aplikace tyto stránky navštěvuje v náhodném pořadí a pro simulaci chování běžného uživatele dále počítá s proměnnými časovými prodlevami, které vkládá mezi návštěvy jednotlivých stránek. Další možnost napodobení chování běžného uživatele souvisela

s předchozí analýzou obsahu HTML dokumentů, které tyto webové stránky obsahují. Díky této analýze byly webové stránky dále přiřazeny do skupin, kde každá tato skupina reprezentuje jisté logické chování uživatele, a to zejména z hlediska návaznosti obsahu jednotlivých webových stránek na sebe.

Mezi možnosti, jak lze vytvořenou aplikaci dále upravit či optimalizovat proces generování síťového provozu, může patřit například zahrnutí většího počtu webových stránek, se kterými se provádí interakce, dále optimalizace algoritmu pro výpočet časové prodlevy mezi navštívením jednotlivých stránek nebo sestavení pořadí, ve kterém se síťový provoz generuje, na základě zcela jiného algoritmu. Na základě vstupního seznamu webových stránek lze dále implementovat možnost výběru objemu síťového provozu, který se má generovat nebo výběr na základě uživatelských profilů.

Alternativním řešením této problematiky může být zejména zcela jiný způsob shromáždění vstupních dat, respektive odkazů na různé webové stránky, neboť při velkém objemu těchto vstupních dat se manuální práce s HTML obsahem jednotlivých stránek jeví jako značně nepraktická.

Co se týče jednotlivých serverů, se kterými aplikace komunikuje prostřednictvím předem stanovených protokolů, nabízí se z hlediska generování síťového provozu především možnosti dalších interakcí, odvíjejících se od charakteristik daných protokolů. Pro aplikaci se samozřejmě nabízí možnost implementovat klientské části i pro další protokoly, nicméně tato volba by vyžadovala rovněž konfiguraci odpovídajících serverů tak, aby měly zajištěno ověřování uživatelů vůči stejnému LDAP serveru, pro který by byly konkrétní přihlašovací údaje platné.

Použitá literatura

[1] ANDREA, Harris. *10 Best Network Traffic Packet Generator Software Tools (in 2020). Networks Training* [online]. [cit. 2021-01-09]. Dostupné z: https://www.networkstraining.com/best-network-traffic-packet-generator-tools/

[2] TECHOPEDIA. *What is Network Traffic? - Definition from Techopedia* [online]. [cit. 2021-01-09]. Dostupné z: https://www.techopedia.com/definition/29917/network-traffic

[3] SHAFIQ Muhammad, Xiangzhan YU, LAGHARI Asir Ali, Lu YAO, KARN Nabir Kumar a ABDESSAMIA Foudil. *Network Traffic Classification Techniques and Comparative Analysis Using Machine Learning Algorithms* [online]. Harbin Institute of Technology, 2019 [cit. 2021-01-09]. Dostupné z: https://www.researchgate.net/publication/316900016_Network_Traffic_Classification_techniques_and_comparative_analysis_using_Machine_Learning_algorithms

[4] MATHWORKS. *What Is Machine Learning?* [online]. [cit. 2021-01-09]. Dostupné z: https://www.mathworks.com/discovery/machine-learning.html

[5] MAH, Bruce A. *An Empirical Model of HTTP Network Traffic* [online]. Proceedings of INFOCOM '97. Kobe, Japonsko, 1997 [cit. 2021-01-09]. Dostupné z: http://www.kitchenlab.org/www/bmah/Papers/Http-Infocom.pdf

[6] DNSSTUFF. *Best 10 Packet Sniffer and Capture Tools: Comparsion and Tips* [online]. 2019 [cit. 2021-01-09]. Dostupné z: https://www.dnsstuff.com/packet-sniffers

[7] FS. *Port Mirroring Explained: Basis, Configuration & FAQs* [online]. 2017 [cit. 2021-04-25]. Dostupné z: https://community.fs.com/blog/port-mirroring-explained-basis-configuration-and-fa-qs.html

[8] VAŇKOVÁ, Jana a Michal ČERNÝ. *Parametry počítačových sítí* [online]. 2012 [cit. 2021-01-09]. Dostupné z: https://clanky.rvp.cz/clanek/a/13991/15061/PARAMETRY-POCITACOVYCH-SITI.html/

[9] ZVONÍČEK, Josef. *Přenosová média* [online]. [cit. 2021-01-09]. Dostupné z: http://pepa.zvonicek.info/inf/prenosova_media.html

[10] GRYGÁREK, Petr. *Quality of Service (QoS) v IP sítích* [online]. Ostrava [cit. 2021-01-09]. Dostupné z: http://www.cs.vsb.cz/grygarek/SPS/lect/QoS/QoS.html. Vysoká škola báňská - Technická univerzita Ostrava.

[11] PARKER, Jeff. *What is Qos or Quality of Service?* [online]. 2020 [cit. 2021-01-09]. Dostupné z: https://www.pcwdld.com/what-is-qos

[12] SPEEDCHECK. *What is Jitter?* [online]. [cit. 2021-01-09]. Dostupné z: https://www.speedcheck.org/wiki/jitter/

[13] TECHTARGET. *What is OSI Model? Definition from WhatIs.com* [online]. 2019 [cit. 2021-01-09]. Dostupné z: https://searchnetworking.techtarget.com/definition/OSI

[14] MACHNÍK, Petr a MICHALEK Libor. *Základy telekomunikačních sítí pro integrovanou výuku VUT a VŠB-TUO* [online]. Ostrava, 2017 [cit. 2021-01-09]. Dostupné z: https://lms.vsb.cz/pluginfile.php/581966/mod_resource/content/7/Telekomunika%C4%8Dn%C3%AD%20s%C3%ADt%C4%9B%20%281.%20%C4%8D%C3%A1st%29.pdf. Vysoká škola báňská - Technická univerzita Ostrava.

[15] BEAL, Vangie. *The 7 Layers Of The OSI Model* [online]. 2019 [cit. 2021-01-09]. Dostupné z: https://www.webopedia.com/quick_ref/OSI_Layers.asp

[16] JAROŠ, Ondřej. *Implementace pokročilého generátoru síťového provozu* [online]. Ostrava, 2014 [cit. 2021-01-09]. Dostupné z: http://hdl.handle.net/10084/103952. Bakalářská práce. Vysoká škola báňská - Technická univerzita Ostrava.

[17] FS. *TCP/IP vs. OSI: What's the Difference Between the Two Models?* [online]. 2017 [cit. 2021-01-09]. Dostupné z: https://community.fs.com/blog/tcpip-vs-osi-whats-the-difference-between-the-two-models.html

[18] TECHOPEDIA. *What is Android SDK? - Definition from Techopedia* [online]. [cit. 2021-01-11]. Dostupné z: https://www.techopedia.com/definition/4220/android-sdk

[19] GARGENTA, Marko a NAKAMURA Masumi. *Android Overview - Learning Android, 2nd Edition* [online]. [cit. 2021-01-11]. Dostupné z: https://www.oreilly.com/library/view/learning-android-2nd/9781449336226/ch01.html

[20] ANDROID DEVELOPERS. *App Widgets Overview* [online]. [cit. 2021-01-11]. Dostupné z: https://developer.android.com/guide/topics/appwidgets/overview

[21] VRBACKÝ, Jakub. *Baterie telefonu, aneb co musíte vědět a co nesmíte dělat (vědecké okénko)* [online]. 2012 [cit. 2021-01-11]. Dostupné z: https://mobilizujeme.cz/clanky/baterie-telefonu-aneb-co-musite-vedet-a-co-nesmite-delat-vedecke-okenko

[22] MARTIN, Taylor. *The truth about Android task killers and why you don't need them* [online]. 2011 [cit. 2021-01-11]. Dostupné z: https://www.phonedog.com/2011/06/26/the-truth-about-android-task-killers-and-why-you-don-t-need-them

[23] MEIER, Reto. *Professional Android 4 Application Development* [online]. Wrox, 2012 [cit. 2021-01-11]. ISBN 978-1-118-10227-5. Dostupné z: https://books.google.cz/books?id=g3hAdK1IBkYC&pg=PT53&redir_esc=y#v=onepage&q&f=false

[24] CALLAHAM, John. *The history of Android: The evolution of the biggest mobile OS in the world* [online]. 2020 [cit. 2021-01-11]. Dostupné z: https://www.androidauthority.com/history-android-os-name-789433/

[25] MARVAN, Filip. *Mobilní operační systém Android: Jak to všechno začalo* [online]. 2011 [cit. 2021-01-11]. Dostupné z: https://diit.cz/clanek/mobilni-operacni-system-android

[26] ANDROID OPEN SOURCE PROJECT. *Brand Guidelines* [online]. [cit. 2021-01-11]. Dostupné z: https://source.android.com/setup/start/brands

[27] MENON, Murali K. *Android Nougat: Here's why Google names the OS after sweets* [online]. Mumbai, 2016 [cit. 2021-01-11]. Dostupné z: https://indianexpress.com/article/lifestyle/food-wine/from-donut-to-nougat-why-are-android-versions-named-after-sweets-2887237/

[28] ANDROID DEVELOPERS. *Platform Architecture* [online]. [cit. 2021-01-11]. Dostupné z: https://developer.android.com/guide/platform

[29] GOEL, Deepam. *Understanding Android Architecture* [online]. 2018 [cit. 2021-01-11]. Dostupné z: https://medium.com/@deepamgoel/understanding-android-architecture-1f0fb4b52f90

[30] ANDROID OPEN SOURCE PROJECT. *Kernel* [online]. [cit. 2021-01-11]. Dostupné z: https://source.android.com/devices/architecture/kernel

[31] ANDROID ENTUSIASTS. *Which Android runs which Linux kernel?* [online]. 2013, 2020 [cit. 2021-01-11]. Dostupné z: https://android.stackexchange.com/questions/51651/which-android-runs-which-linux-kernel

[32] HENOTE TECHNOLOGIES. *Top 5 Popular Programming Languages for Mobile App Development* [online]. 2019 [cit. 2021-01-12]. Dostupné z: https://medium.com/@Henote/top-5-popular-programming-languages-for-mobile-app-development-d55d2ba368b3

[33] HAIJE, Erin Gilliam. *Top 20 Mobile Development Tools: An Overview* [online]. 2019 [cit. 2021-01-12]. Dostupné z: https://mopinion.com/mobile-development-tools-an-overview/

[34] ANDROID DEVELOPERS. *Meet Android Studio* [online]. 2020 [cit. 2021-01-12]. Dostupné z: https://developer.android.com/studio/intro

[35] ANDROID DEVELOPERS. *App Manifest Overview* [online]. 2020 [cit. 2021-01-12]. Dostupné z: https://developer.android.com/guide/topics/manifest/manifest-intro

[36] ANDROID OPEN SOURCE PROJECT. *Codenames, Tags, and Build Numbers* [online]. 2021 [cit. 2021-01-12]. Dostupné z: https://source.android.com/setup/start/build-numbers

[37] EMPEY, Charlotte. *Co je VPN a jak funguje? Váš základní průvodce.* [online]. 2019-04-01 [cit. 2021-03-01]. Dostupné z: https://blog.avast.com/cs/co-je-vpn-a-jak-funguje

[38] MOCAN, Tim. *What Is OpenVPN & How Does OpenVPN Work?* [online]. 2019-02-01 [cit. 2021-03-01]. Dostupné z: https://www.cactusvpn.com/beginners-guide-to-vpn/what-is-openvpn/

[39] *Lightweight Directory Access Protocol (LDAP)* [online]. 2019-06-21 [cit. 2021-03-01]. Dostupné z: https://www.geeksforgeeks.org/lightweight-directory-access-protocol-ldap/

[40] NEVLUD, Pavel. VŠB-TECHNICKÁ UNIVERZITA OSTRAVA. *Komunikační sítě II pro integrovanou výuku VUT a VŠB-TUO* [online]. Ostrava, 2014 [cit. 2021-03-01]. Dostupné z: https://lms.vsb.cz/pluginfile.php/1227169/mod_resource/content/1/pksII.pdf

[41] SOBERS, Rob. *The Difference Between Active Directory and LDAP* [online]. [cit. 2021-03-01]. Dostupné z: https://www.varonis.com/blog/the-difference-between-active-directory-and-ldap/

[42] *Directory Services* [online]. [cit. 2021-03-01]. Dostupné z: https://www.managedsolution.com/directory-services/

[43] BŘICHÁČEK, Zdeněk. *LDAP adresářové servery pro správu uživatelských identit* [online]. Bankovní institut vysoká škola Praha, 2009 [cit. 2021-03-01]. Dostupné z: https://is.ambis.cz/th/lwerq/LDAP_adresarove_servery_pro_spravu_uzivatelskych_identit.pdf

[44] *The Java CIFS Client Library* [online]. 2017-12-21. Dostupné z: https://www.jcifs.org/

[45] *Apache Commons* [online]. 2020-08-07. Dostupné z: https://commons.apache.org/proper/commons-net/download_net.cgi

[46] *ownCloud: Open source řešení pro ukládání dat v cloudu* [online]. 2018-05-02 [cit. 2021-04-10]. Dostupné z: https://www.linuxexpres.cz/software/owncloud-open-source-reseni-pro-ukladani-dat-v-cloudu

[47] SCARPATI, Jessica. *What is Session Initiation Protocol (SIP)? A definition from WhatIs.com.* [online]. 2019-06. [cit. 2021-04-10]. Dostupné z: https://searchunifiedcommunications.techtarget.com/definition/Session-Initiation-Protocol

[48] TUTORIALSPOINT. *Session Initiation Protocol – Introduction* [online]. [cit. 2021-04-10]. Dostupné z: https://www.tutorialspoint.com/session_initiation_protocol/session_initiation_protocol_introduction.htm

[49] *Email Protocols – POP3, SMTP and IMAP Tutorial* [online]. [cit. 2021-04-18]. Dostupné z: https://www.siteground.com/tutorials/email/protocols-pop3-smtp-imap/

[50] *Postfix SASL Howto* [online]. [cit. 2021-04-18]. Dostupné z: http://www.postfix.org/SASL_README.html

[51] *javamail-android* [online]. Dostupné z: https://code.google.com/archive/p/javamail-android/

[52] DNSSTUFF. *Best Network Traffic Generator and Simulator Stress Test Tools* [online]. 2019 [cit. 2021-04-21]. Dostupné z: https://www.dnsstuff.com/network-traffic-generator-software

Seznam příloh

Příloha A: ***Dostupná v IS Edison***

Obsah:

- Adresář, obsahující aplikaci pro generování síťového provozu, vyvíjenou v programu Android Studio 4.1.2.